The Anatomy of Design: Foundations, Models and Applications

van Langen, P.H.G.

2002

document version
Publisher's PDF, also known as Version of record

The Anatomy of Design

Foundations, Models and Applications

SIKS Dissertation Series No. 2002-16.

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Langen, Pieter H.G. van, 1966 –
The Anatomy of Design: Foundations, Models and Applications
ISBN 90-9016375-1
NUGI 984 (Artificial Intelligence)

Thesis Vrije Universiteit Amsterdam. – Illustrated, with references and summary in Dutch.
Cover design by the author.

Printed by Febodruk B.V., The Netherlands.

VRIJE UNIVERSITEIT

The Anatomy of Design: Foundations, Models and Applications

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op vrijdag 22 november 2002 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

PETRUS HERMANUS GERARDUS VAN LANGEN

geboren te Heerhugowaard

promotoren: prof.dr. F.M.T. Brazier
prof.dr. J. Treur

Preface

This thesis is the result of a research project on Artificial Intelligence in Design that started in 1992. Working as a contract researcher in the Department of Artificial Intelligence at the Vrije Universiteit Amsterdam, I became interested in design as a topic of research. The first visible result was a paper about modelling design processes in a formal manner, presented at the IJCAI 1993 workshop on AI in Design. Many papers about the foundations, models and applications of design followed, in workshops, conferences, journals, and a book.

My interest in design is founded in my childhood. Living in the country, I spent many hours creating small-scale models of tractors and agricultural machines. I also liked to design and build toy versions of motorcycles, race cars, trucks, and aircraft—in short, things that move. Therefore, with hindsight, it is not surprising that this thesis is about design.

Also in my current job as a knowledge engineer, modelling and designing is what I like to do most of all. Although working full-time outside the university has surely slowed down the process of writing this thesis, it has given me the opportunity to put my research ideas to the test in practice. This has led to a number of improvements, in this thesis as well as in my profession. Since 1996, I have been developing and implementing knowledge models, knowledge-intensive applications, knowledge infrastructures, and knowledge management strategies in a completely different setting.

I am indebted to many people, all whom have provided assistance, guidance, constructive criticism, or any other form of support during the preparation of this thesis. Without their support, this thesis would not have been possible.

First and foremost, I would like to thank my supervisors, Frances Brazier and Jan Treur. Ever since the start, they have been a source of inspiration, good advice, encouragement, and enthusiasm. Doing research and writing papers together has been a pleasant and instructive experience, and has given me a great deal of insight into the world of science.

I also would like to thank the members of the reading committee: Paul de Bra, John Gero, Frank van Harmelen, John-Jules Meyer, and Nel Wognum. They were all willing to read this voluminous thesis and prepare for the thesis defence at relatively short notice.

Furthermore, I would like to thank my non-official proof readers: Henk Diepenmaat, Ger Haan, Loek Schoenmaker, Mark Willems, and Niek Wijngaards. A special thanks to Niek, with whom I have had many in-depth discussions about design and re-design during the years of research. Also a special thanks to Mark, since his thesis gave me the idea for the main title of this thesis.

I would like to express my gratitude to the researchers with whom I wrote the papers and reports that underlie this thesis: Frances Brazier, Henk Diepenmaat, David de Klerk, Tinus Pulles, Zsofia Ruttkay, Jan Treur, Mark Willems, and Niek Wijngaards.

I would also like to express my gratitude to the Artificial Intelligence Department of the Vrije Universiteit, and the rest of the staff at the Faculty of Sciences (and the former Faculty of Mathematics and Computer Science). A special thanks to the room mates that I have had over the years: Frank Cornelissen, Ioa Gavrila, Onno Kubbe, Loek Schoenmaker, Sander van Splunter, Wieke de Vries, Fred Wan, Mark Willems, and Niek Wijngaards.

Furthermore, I would like to express my gratitude to the researchers from the SIKS community (the Dutch Research School for Information and Knowledge Systems), and to researchers from the international Artificial Intelligence in Design research community, whom I have met at various conferences in different parts of the world. I am certain that the papers and presentations of renowned researchers from these communities, as well the discussions I have had with them, have been a great source of inspiration for this thesis.

Part of my research has been made possible by third parties. I would like to thank the Dutch Foundation for Knowledge-Based Systems (SKBS) and the Netherlands Organisation for Applied Scientific Research (TNO). SKBS and TNO co-sponsored and created room for a research project, on which Chapter 11 of this thesis is based. I also would like to thank Frans van der Walle, who employed me in 1995 as a systems designer in his company Novoware, and gave me one task, namely to continue my research.

Furthermore, I would like to thank the many current colleagues who have shown great interest in my research. A special thanks to my fellow knowledge engineers, who have eagerly waited for this thesis to appear ever since they have known of its existence.

Last but surely not least of all, I would like to thank my parents, Gerard and Martha van Langen, and my family and friends, for their enduring support and interest. In particular, I would like to thank my brother Herman van Langen and my friend Mariëtte Kuijs for their assistance during the defence of this thesis.

Contents

Part IV: Design Applications

Part V: Design Themes

Part VI: Main Contributions and Directions for AI in Design Research

Appendixes

—

Chapter 1

The Anatomy of Design

This chapter introduces design as the main topic of this thesis. It proposes to investigate the anatomy of design as a process, in order to better understand practical design processes and to develop useful design support systems. The chapter briefly reviews the main results of design research, concluding that such an investigation still offers a challenge. It describes the approach of our research, which is to analyse design processes at the knowledge level, using logic as a means. Finally, the chapter provides an outline of this thesis.

Our world today would be unthinkable without *artefacts.* Both in personal life and business, we have filled our environment with a many (artificial) objects for different kinds of use. Whether such objects are physical (such as houses) or non-physical (such as computer software), they have one or more *functions* that we attribute to them. That is, we find that they have a specific utility, such as providing a place to live or to enable electronic stock-trading.

To create an artefact for a given purpose may be a true challenge. For example, people have wanted to fly for ages, but it took until A.D. 1903 before the Wright brothers managed to construct an aircraft that stood the test. In general, the purposeful creation of an artefact demands thought about what exactly the *requirements* are and what *structure* or *form* the artefact must have to behave in the required way. This setting is typical for a *design activity*.

A design activity is usually referred to in terms typically associated with a specific type of application domain. For example, *architecture* refers to the design of buildings, *mechanical engineering* to the design and construction of instruments (i.e., tools, equipment and machines), and *industrial design* to the design of industrial products (e.g., in terms of shape, texture and colour). Nowadays, design is seen as more than just a business activity: it is understood to represent a variety of cultural activities—policies, institutions and human behaviour itself, for instance, are increasingly thought of as objects of design ([Schön, 1983]).

With the introduction of computers, automated systems have been developed to support professional designers. Since the 1980s, architects, building constructors, industrial designers, and electrical, mechanical and civil engineers increasingly make use of Computer Aided Design systems. These systems provide facilities for drawing artefacts and three-dimensional visualisation and sometimes also simple constraint checking (e.g., testing whether or not two walls forming a corner are really connected) and simulation (e.g., a finite element analysis to assess the structural properties of a ship's hull). With the increasing stress on shortening the time-to-market (i.e., the time it takes to turn a business opportunity into a product that can be sold on the market), companies have realised that it is important to pay attention to how they can best design their products.

In practice, designing generally proves to be a difficult task: a designer is confronted with 'problematic situations which are puzzling, troubling and uncertain' ([Schön, 1983]). Most design researchers agree that the difficulty arises from the fact that the needs and desires to be addressed by a designer, as well as the initial design problem that a designer derives from these needs and desires, are *ill structured* (a term introduced by Simon [Simon, 1973]). This means that, as a *whole*, such a design problem cannot be subsumed under a familiar category: it is unique in terms of the function that the artefact to be designed has to fulfil, the real-world environment in which the artefact is meant to fulfil this function, or both. (Several researchers in different wordings have expressed this view; see, for instance, [Alexander, 1964; Simon, 1973; Schön, 1983]). As a consequence, available methods and techniques for known problem categories fail to effectively solve such a design problem as a whole.

Since the 1960s, design is a subject of active scientific study. Within and across different disciplines, there is a continual exchange of ideas between designers, design researchers and developers of automated systems for design and design support. One area in which different disciplines come together is that of Artificial Intelligence in Design (often abbreviated AI in Design), which relates to Artificial Intelligence, Computer Science, Logic, Human-Computer Interaction, Architecture, Mechanical Engineering and many other fields.

This thesis intends to contribute to Artificial Intelligence in Design research by providing a basis for a better understanding of practical design processes and the development of design support systems. A designer with a better understanding of design processes, and supported by useful tools, is hopefully in a better position to improve his or her performance. For this contribution, techniques from Artificial Intelligence and Logic are used.

The central hypothesis of this thesis is that, although (initial) design problems are generally ill structured, the main structure of design as a process is well defined and even generic. That is, a design process has an *anatomy* that is independent of its application domain, the design problem at hand and the repertory of available design methods and techniques.

This chapter is organised as follows. Section 1.1 introduces the anatomy of design as a process and explains the central hypothesis of this thesis. Section 1.2 briefly reviews results of related design research, and Section 1.3 describes the approach of our research. Finally, Section 1.4 provides an outline of this thesis.

1.1 Design as a Process

This thesis takes the stance that, as design problems are ill structured, designing cannot be defined as making a description of an artefact to be created, on the basis of a design problem stating perfectly the function that the artefact has to fulfil and the environment in which the artefact is meant to fulfil this function. In practice, designing involves determining what the design problem to be solved is in order to fulfil the needs and desires of a client, which happens before or (mostly) during development of the design artefact description. This view underlies the theory and models of design and their applications presented in this thesis, and is shared among several researchers; a few examples are discussed below.

Schön states that, as a design problem does not present itself in a well defined structure, a designer has to impose such a structure on it before it can be effectively solved [Schön, 1983]. In his view, design involves problem solving and, most importantly, problem setting: the designer must 'make sense of an uncertain situation that initially makes no sense.'

Bijl defines design as a process of synthesis, with design objects that can be described and perceived in many different ways, conflicting criteria for validating results, many solutions, and based on design knowledge that is informal, incomplete and partly intuitive [Bijl, 1987]. He states that, rather than a problem-solving process, design can be better described as 'an activity of event exploration, in which partial responses lead to redefinition of a goal.'

Treur considers design as a process of constructing an explicit description of the structure of an artefact, such that a given list of design requirements is satisfied and such that the artefact can actually be realised; however, when it turns out during the design process that there does not exist such an description, some of the original design requirements may have to be changed [Treur, 1989]. He states that, consequently, it is necessary in design processes to reason not only *from* design requirements (using design requirements as conditions in logical inferences, where the object of reasoning is the artefact being designed) but also *about* design requirements (using design requirements as objects of reasoning in logical inferences).

Gero characterises design as a goal-oriented, constrained, decision-making, exploration and learning activity that operates within a context that depends on the designer's perception [Gero, 1990]. In his opinion, exploration, learning and context make design different from other decision-making activities. During exploration, the designer learns about intended, emerging (i.e., observed but unintended) or unsatisfactory features of the artefact being designed, which changes his/her perception of the design problem and its possible solutions.

Smithers, Corne and Ross consider design problems as typically ill structured: that is, initial requirement descriptions are typically inconsistent, ambiguous, imprecise, or incomplete [Smithers, Corne and Ross, 1994]. They view design as an explorative process, with the important characteristic that a consistent, unambiguous, precise, and complete requirements description is developed along with a design solution that satisfies it. In other words, design is not a transformation of a well defined design problem into a design solution, but a process of exploring possible ways to structure the design problem and solve it.

Protzen, Harris and Cavallin go a step further by stating that design problems are not ill structured, but even worse: they are wicked problems [Protzen, Harris and Cavallin, 2000]. They claim that design problems do not have a definite or exhaustive formulation and therefore should be classified as wicked problems (a term introduced by Rittel and Webber [Rittel and Webber, 1969]). However, Protzen *et al.*'s objections against ill-structuredness as a defining characteristic of design problems are not so much based on Simon's definition of ill structured problems (viz., those problems that are not well defined). Their arguments are based on Simon's arguable interpretation of his own definition, which is that an ill structured problem is a problem of which the *equivalent* well defined formulation has not yet been discovered [Simon, 1973]. If this interpretation is singled out, there is no reason to distinguish wicked problems from ill structured problems. Hence, this thesis will use the term ill-structuredness, which is in line with what the majority of the AI in Design community does.

Given the ill-structuredness of design problems, which accounts for the view of design as exploration, the question as to the structural properties of design processes arises. This thesis takes the stance that, despite the nature of design problems, design processes need not be ill structured. The central hypothesis is that *a design process has a well defined and generic main structure, or* anatomy*, that is independent of the application domain, the design problem at hand and the repertory of available design methods and techniques.* Investigating this anatomy contributes to a better understanding of design processes as well as to the development of useful design support systems.

1.2 State of the Art of Design Research

This section describes the state of the art of design research from a helicopter view. It is not intended to analyse what can be learned from the results already achieved in the past, which is the purpose of Chapter 2. Rather, this section provides a small survey of the main areas and topics of design research in the past four decades, with particular emphasis on AI in Design research.

The survey describes typical subjects of design research, scientific disciplines underlying most of the research, and typical domains to which the results apply. Many different topics are investigated in design research, such as:

- models of design problem spaces (especially operational models of design problems),
- models of design processes,
- design representation (i.e., representation of artefacts, environments, design problems, and design knowledge, including design prototypes),
- design methods (sometimes formulated as problem solving methods),
- techniques for (knowledge-based) reasoning in design,
- design support systems and automated design systems,

- design themes such as design requirements engineering, design for manufacturing (or maintenance, recyclability, etc.), management of conflicts in design, design strategies, design rationale, creativity in design, design history, and learning in design.

An upcoming subject is distributed design, where two or more agents with identical or different design capabilities co-operate and co-ordinate their efforts to bring one or more design projects to a successful end.

There are more design process models described in research literature than design theories. Design process models are less complex and powerful and have a different purpose than design (process) theories. Their advantage is that they can be used fairly directly as a basis for developing computational means to support designers. However, design process models are based on many assumptions about the design process, which often remain tacit and therefore hinder a better understanding of design processes. Relatively few papers attempt to provide a theoretical basis for design problem spaces, design solution spaces, functional mappings between these spaces and design processes; notwithstanding their value, none of the current theories is acknowledged to be completely satisfactory.

Design research makes use of many scientific disciplines. Typical disciplines underlying the research are:

- Decision Theory,
- Mathematics (e.g., Set Theory and Graph Theory),
- Logic,
- Information Processing Theory,
- Cognitive Psychology,
- Software Engineering (including Human-Computer Interaction),
- Artificial Intelligence, with methodologies and techniques such as constraint satisfaction, genetic algorithms, rule-based reasoning, case-based reasoning, truth maintenance, meta-level reasoning (using reflection principles to shift between two or more levels of reasoning) and generative reasoning (on the basis of grammars).

Despite the many disciplines, papers usually focus on a single discipline and rarely on two or more of them.

Frequently cited application domains for the results of design research are engineering (most often mechanical engineering and product engineering) and architecture. However, the AI in Design community intends a broader applicability of its research results and usually makes little or no assumptions about the application domain.

This survey shows that there are many topics to be researched and scientific disciplines on which to base the research. Given that none of the existing design theories and models are satisfactory for the whole range (see Chapter 2), it can be concluded that an in-depth investigation of the anatomy of design as a process still offers a challenge for research.

1.3 Logical Analysis of Design Processes

The approach of our research on the anatomy of design as a process can be outlined as analysis of design processes at the knowledge level, using logic as a means for analysis and with frequent interactions between theory and practice. This section explains and motivates the approach.

Analysing design as a process means that the structure of a design process is determined by observing and deriving the static aspects and dynamic aspects of the design process and its sub-processes. Such an analysis results in an overview of concepts and logical relationships between concepts.

The *static aspects* concern the possible descriptions involved in a design process, such as sets of design requirements, or design object descriptions. The logical relationships between these types of descriptions include, for example, consistency of design object descriptions, and satisfaction of a design requirement by a design object description.

The *dynamic aspects* concern the possible states and sequences of state transitions within a design process, such as modification of a set of design requirements, or deductive refinement of a design object description. The logical relationships include, for example, the notion of a design solution.

Together, the static aspects and dynamic aspects form a basis for a declarative (i.e., non-procedural) and representation-independent description of the characteristics of design processes. This is a *knowledge-level* description ([Newell, 1982]), as it describes that:

- the medium to be processed is *knowledge* (that can be specified in the form of a functional relation between the input and output of sub-processes),
- there are components that provide processing capabilities in terms of *goals*, *actions* and *bodies* (in the form of sub-processes),
- there are *laws of composition* that permit components to be assembled into systems (in the form of the composition of a process by means of sub-processes and exchange of information between sub-processes), and
- the law of behaviour that determines how system behaviour depends on the component behaviour and the structure of the system is the *principle of rationality*, which means that actions of a design process are selected to attain goals of the design process (in the form of the functional relation between activation states of a process on the one hand and activation states of the sub-processes as well as exchanges of information between the sub-processes on the other hand).

To express the static and dynamic aspects of design processes, a language is needed with a clear semantics. By many researchers, *logic* is deemed appropriate for analysing the knowledge level: it is declarative and does not enforce a particular representation ([Newell, 1982]). This does not imply that logic is also the preferred representation to use for a given applica-

tion domain. It merely forms a solid basis for analysis of design processes and for the selection of an appropriate representation language for a specific application domain.

The final point to be made in this section is our approach to develop a logical theory of design by having practice and theory interact frequently. This approach is based on the fact that design is a cognitive process of which the observable aspects (making notes, gesturing, sketching, etc.) may reveal only part of the structure. Therefore, a theory of design cannot be proven to be correct and complete in a mathematical sense; rather, it is at best consistent with the known results from empirical studies.

Our logical theory of design has been developed and improved by regularly studying design literature and by an iterative theory development process of observing design processes in practice, developing a new version of the theory and putting the new version to the test by conducting experiments in practical application domains. In this way, our logical theory of design has converged gradually into a robust version, which stands the test in many practical situations.

1.4 Outline of the Thesis

This thesis is divided into six parts, which together consist of sixteen chapters. Figure 1.1 shows a schematic overview of the thesis, in terms of these parts and their relations.

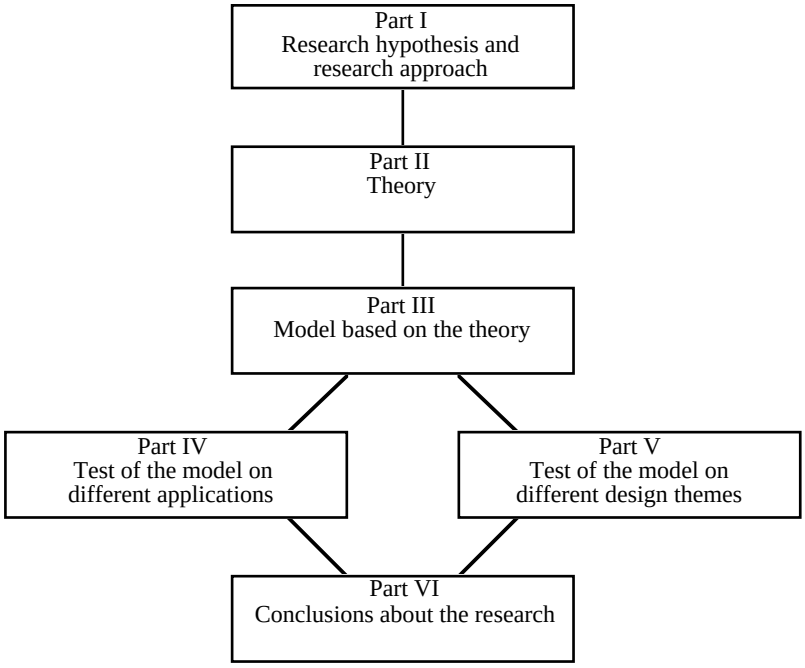


FIGURE 1.1. Schematic overview of the thesis.

Part I: A proposal for AI in Design research. Part I describes the central hypothesis of the research presented in this thesis, and the research approach that has been followed. Chapter 2 reviews the main results of related design research concerning design theories and models, methods and support systems.

Part II: Foundations of design. Part II presents a logical theory of design, which describes the anatomy of design as a process. That is, (first-order predicate) logic has been used as a vehicle to describe the structure of design processes. By formulating the theory at the knowledge level, it can be interpreted independent of procedural and representational issues.

Chapter 3 defines the static aspects of design, which concern the possible descriptions that are involved in a design process. The main descriptions introduced are *requirement qualification set* and *design object description*, and the main logical relationships are assessment relations, such as *satisfaction* of design requirement by a design object description.

Chapter 4 defines the dynamic aspects of design, which concern the possible states and sequences of state transitions within a design process. Design process states contain design information that allows to reason at different *reflection levels*. Chapter 4 defines the object level, as well as the first, second, and third meta-level of reasoning within a design process.

Part III: A generic model of design. Part III presents a generic design model, called GDM, which is based on the logical theory presented in Part II. GDM offers a specification, which can be directly translated to computer software. It is claimed to be useful as a basis for analysing complex, practical design processes and for developing design support systems.

Chapter 5 presents an overview of DESIRE, a compositional development method for modelling complex reasoning tasks (such as design tasks) and agent systems (such as mobile design agents on the Internet). DESIRE defines a knowledge-level logical language, suitable to model the static and dynamic aspects of design processes, and therefore it has been used to develop GDM.

Chapter 6 presents the two upper levels of the three process abstraction levels of which GDM consists. It describes the design process (which is at the top level) and its three sub-processes: the manipulation of requirement qualification sets (i.e., sets of requirements as well as qualified requirements expressing satisfaction conditions on the requirements), the manipulation of design object descriptions and the overall co-ordination of the design process. The chapter describes the input and output of each of these processes, as well as the way these processes exchange information.

Chapter 7 describes GDM on the third process abstraction level, detailing the sub-process of manipulating requirement qualification sets. This sub-process consists in total of four sub-processes.

Chapter 8 also describes GDM on the third process abstraction level, detailing the sub-process of manipulating design object descriptions. Also this sub-process consists of four sub-processes.

Part IV: Design applications. Part IV presents a specialisation of GDM and two design applications based on this specialisation. Both practical applications have been made to empirically test the validity of the logical theory of design, as well as the usefulness of GDM for analysing complex, practical design processes and developing design support systems.

Chapter 9 presents a specialisation of GDM, which consists of an elaboration of requirement qualification set modification and of design object description modification. This specialisation has turned out to be useful in many practical design applications.

Chapter 10 describes an application of GDM to an elevator configuration task called VT. VT is a special kind of design task, where the design object to be configured is an elevator, and where the design requirements consist of customer specifications, building dimensions and constraints.

Chapter 11 shows how GDM has been used to create a model of environmental inventory. The object to be designed is a process model by means of which the environmental impact of a specific real-life industrial process can be determined, and the design requirements are general conditions on the quality of that process model.

Part V: Design themes. Part V explores three design themes that are regular subjects of research in the AI in Design community: strategic design knowledge, design rationale and conflict management in design. A design theme addresses specific aspects of design processes, and this part demonstrates the usability of GDM in modelling such aspects of design processes.

Chapter 12 focuses on the role of strategic knowledge in design. In an interactive design process, a designer and a design support system interact about the strategy for the design process. To support interactive design processes, a design support system has to be capable of reasoning with strategic knowledge corresponding to the types of strategic interaction that the system may have with the designer. The chapter explains three types of strategic interaction: interaction about the design process, about the manipulation of requirement qualification sets, and about the manipulation of design object descriptions. On the basis of an example, it is shown how GDM can be used to model the corresponding strategic knowledge.

Chapter 13 focuses on the role of history and rationale in design processes. It states that design support systems should make use of design history and design rationale, and describes how GDM can be used to model different types of design rationale. A sample application domain (aircraft design) is used to illustrate the use of design history and design rationale.

Chapter 14 states that conflicts such as contradictions between requirements are inherent to design. The view underlying *conflict management in design* is that design is a managed process of detecting, prioritising, and resolving design conflicts. The complex reasoning processes involved in conflict management in design are highly dynamic and non-monotonic: the resolution of one conflict often results in the creation of another. Chapter 14 analyses possible types of design conflicts and it presents excerpts from a specialisation of GDM that models conflict management in design.

Part VI: Main contributions and directions for AI in Design research. Part VI forms the end of this thesis, with conclusions about what has been achieved in the research and what is still to be done. Chapter 15 describes contributions and limitations of the research, and Chapter 16 presents directions for further research with respect to design theories and models, design methods, and design support systems.

Chapter 2

Related Design Research

This chapter reviews the main results of related design research on theories and models of design, design methods and design support systems. This review concentrates on what can be learned from past research that can be used to the advantage of a logical analysis of the structure of design processes.

The central hypothesis of this thesis is that a design process has a well defined and generic main structure: an anatomy that is independent of the application domain, the design problem at hand and the repertory of available design methods and techniques. Investigating this anatomy by means of a logical analysis contributes to a better understanding of design processes and therefore to the development of a theory of design as a process, and the development of useful design support systems.

This chapter focuses on what can be learned from related design research on theories and models of design, design methods and design support systems. The main emphasis is on identifying those topics that (most) design researchers agree to be relevant in relation to the development of design theories and models, design methods and design support systems.

This chapter is organised as follows. Section 2.1 describes concepts that design theories and models should define, according to design researchers. Section 2.2 describes representations for design methods, and Section 2.3 describes functionality that design support systems should provide. Finally, Section 2.4 describes implications for AI in Design research, in terms of which of the agreed-upon design topics (concepts, representations and functionality) have to be investigated in more depth in order to further a better understanding of design processes and the development of useful design support systems.

2.1 Design Theories and Models

This section describes concepts brought forth by design research (including AI in Design) that are of interest to design theories and design models. A *theory* is a conceptualisation of a universe of discourse, which defines relevant phenomena in that universe of discourse (e.g., objects, states and events), and which explains and predicts relations between these phenomena. To this end, a theory offers definitions that conceptualise phenomena, and axioms that conceptualise relations between phenomena. A *model* may define only part of the relevant phenomena, or it may explain and predict only part of the relations between phenomena in the universe of discourse.[1] Since in design literature the terms 'theory' and 'model' are often used in an informal way, this section does not make a distinction between them.

In design research, often the following concepts and concept relationships are mentioned in relation to a theory or model of design:

- *Object*, or *domain object*
  Something visible or tangible in a domain of application, such as a car, building, or home appliance (e.g., [Akin, 1978; French, 1985; Koller, 1985; Bijl, 1987; Tomiyama and Yoshikawa, 1987; Treur, 1989; Gero, 1990; Alberts, Wognum and Mars, 1992; Wielinga, Akkermans and Schreiber, 1995; Klein, 2000]).

- *Object attribute*
  A qualitative or quantitative characteristic of an individual object, such as the size, shape, or material of a chair (e.g., [Koller, 1985; Bijl, 1987; Tomiyama and Yoshikawa, 1987; Alberts, Wognum and Mars, 1992; Wielinga, Akkermans and Schreiber, 1995; Klein, 2000; Varejão, De Menezes, Garcia, De Souza and Fromherz, 2000]).

- *Object attribute value*, or *object property*
  A specific object's attribute and its value, such as a red Ferrari, or a knife with a blade of four inches (e.g., [Koller, 1985; Bijl, 1987; Tomiyama and Yoshikawa, 1987; Alberts, Wognum and Mars, 1992; Wielinga, Akkermans and Schreiber, 1995; Klein, 2000; Varejão, De Menezes, Garcia, De Souza and Fromherz, 2000]).

- *Object relationship*
  A relationship between objects, such as alignment of the front of two buildings, or compatibility of two systems (e.g., [Akin, 1978; French, 1985; Bijl, 1987; Tong, 1987; Gero, 1990; Alberts, Wognum and Mars, 1992; Smithers, Corne and Ross, 1994; Wielinga, Akkermans and Schreiber, 1995; Klein, 2000]).

[1] A well-known example of the difference between theory and model originates from physics. Currently, there are two *models* of light: a wave model and a particle model. But so far, there is no *theory* of light that explains and predicts the behaviour of light in all circumstances.

- *Object composition*
  A *part-of* relationship between an object and other objects (its parts), such as a tea pot consisting of a jar and a lid (e.g., [Akin, 1978; French, 1985; Koller, 1985; Bijl, 1987; Tong, 1987; Treur, 1989; Gero, 1990; Alberts, Wognum and Mars, 1992; Smithers, Corne and Ross, 1994; Wielinga, Akkermans and Schreiber, 1995; Klein, 2000; Varejão, De Menezes, Garcia, De Souza and Fromherz, 2000]).

- *Domain object knowledge*
  A set of statements about objects, their properties, and their relationships, that hold in a specific application domain (e.g., [Akin, 1978; French, 1985; Bijl, 1987; Tong, 1987; Treur, 1989; Gero, 1990; Alberts, Wognum and Mars, 1992; Brumsen, Pannekeet and Treur, 1992; Smithers, Corne and Ross, 1994; Wielinga, Akkermans and Schreiber, 1995; Smithers, 1998; Klein, 2000]).

- *Context*, or *environment*
  The objects with which a specific object (viz., the design object) is deemed to interact, as well as their properties and their relationships (e.g., [Asimow, 1962; Alexander, 1964; Koller, 1985; Tomiyama and Yoshikawa, 1987; Goel and Pirolli, 1989; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Gero, 1990; Maher, 1990; Smithers, 1998; Klein, 2000]).

- *Object behaviour*
  The way in which an object acts in a given context, such as the behaviour of a car on a slippery road (e.g., [Koller, 1985; Gero, 1990; Alberts, Wognum and Mars, 1992; Klein, 2000; Varejão, De Menezes, Garcia, De Souza and Fromherz, 2000]).

- *Function (of an object)*
  The intended behaviour (i.e., use or purpose) of a specific object, such as fault tolerance of a software system (e.g., [Alexander, 1964; Koller, 1985; Bijl, 1987; Tomiyama and Yoshikawa, 1987; Gero, 1990; Alberts, Wognum and Mars, 1992; Klein, 2000]).

- *Need*, or *desire*
  An often implicit motivation of a human to change the individual or societal situation (e.g., [Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Pugh, 1990; Suh, 1990; Smithers, Corne and Ross, 1994; Varejão, De Menezes, Garcia, De Souza and Fromherz, 2000]).

- *Design requirement*, *design criterion*, or *functional specification*
  An explicit formulation of (part of) a need or desire, concerning the function, behaviour or structure of an object to be constructed (e.g., [Asimow, 1962; Alexander, 1964; French, 1985; Koller, 1985; Mostow, 1985; Tomiyama and Yoshikawa, 1987; Treur,

1989; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Gero, 1990; Pugh, 1990; Suh, 1990; Brumsen, Pannekeet and Treur, 1992; Logan and Smithers, 1992; Smithers, Corne and Ross, 1994; Wielinga, Akkermans and Schreiber, 1995; Smithers, 1998; Klein, 2000; Varejão, De Menezes, Garcia, De Souza and Fromherz, 2000]).

- *Requirement qualification set* (*requirements description* or *design program*)
  A set of design requirements imposed on an object (e.g., [Asimow, 1962; Alexander, 1964; Pahl and Beitz, 1984; French, 1985; Koller, 1985; Tomiyama and Yoshikawa, 1987; Tong, 1987; Goel and Pirolli, 1989; Treur, 1989; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Logan and Smithers, 1992; Smithers, Corne and Ross, 1994; Wielinga, Akkermans and Schreiber, 1995; Smithers, 1998; Klein, 2000]).

- *Requirement qualification set space*
  The space of possible requirement qualification sets for a specific application domain (e.g., [Mostow, 1985; Tong, 1987; Goel and Pirolli, 1989; Wielinga, Akkermans and Schreiber, 1995]).

- *Design object description (design* or *product design specification*)
  A description of the structure or form of a specific object, and a prescription for its construction (e.g., [Asimow, 1962; Alexander, 1964; Pahl and Beitz, 1984; French, 1985; Koller, 1985; Mostow, 1985; Tomiyama and Yoshikawa, 1987; Tong, 1987; Goel and Pirolli, 1989; Treur, 1989; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Gero, 1990; Maher, 1990; Pugh, 1990; Suh, 1990; Logan and Smithers, 1992; Smithers, Corne and Ross, 1994; Wielinga, Akkermans and Schreiber, 1995; Smithers, 1998; Klein, 2000; Varejão, De Menezes, Garcia, De Souza and Fromherz, 2000]).

- *Design object description space*
  The space of possible design object descriptions for a specific application domain (e.g., [Tong, 1987; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Gero, 1990; Logan and Smithers, 1992]).

- *Design activity*, *design event*, *design (sub-)task*, *design operation,* or *design function*
  An act of designing, such as the refinement or structuring of a design problem, and the generation, visual perception, or evaluation of a design solution (e.g., [Asimow, 1962; Archer, 1970; Akin, 1978; French, 1985; Koller, 1985; Bijl, 1987; Tong, 1987; Goel and Pirolli, 1989; Brown and Chandrasekaran, 1989; Goel and Chandrasekaran, 1989; Treur, 1989; Chandrasekaran, 1990; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Strelnikov and Dmitrevich, 1991; Brumsen, Pannekeet and Treur, 1992; Smithers, Corne and Ross, 1994; Wielinga, Akkermans and Schreiber, 1995; Smithers, 1998; Varejão, De Menezes, Garcia, De Souza and Fromherz, 2000]).

- *Design process*
  A sequence of design activities, such as building design, industrial design, civil engineering, or software design (e.g., [Asimow, 1962; Pahl and Beitz, 1984; French, 1985; Koller, 1985; Mostow, 1985; Bijl, 1987; Tomiyama and Yoshikawa, 1987; Tong, 1987; Goel and Pirolli, 1989; Treur, 1989; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Gero, 1990; Maher, 1990; Pugh, 1990; Suh, 1990; Strelnikov and Dmitrevich, 1991; Brumsen, Pannekeet and Treur, 1992; Logan and Smithers, 1992; Smithers, Corne and Ross, 1994; Wielinga, Akkermans and Schreiber, 1995; Smithers, 1998; Klein, 2000; Varejão, De Menezes, Garcia, De Souza and Fromherz, 2000]).

- *Design (process) state*, or *design stage*
  The design information known at a specific moment during a design process (e.g., [Asimow, 1962; Mostow, 1985; Treur, 1989; Brumsen, Pannekeet and Treur, 1992; Smithers, Corne and Ross, 1994; Smithers, 1998; Klein, 2000]).

- *Design step* (*move*, *change*, or *revision*)
  A transition of one design state into another, as a result of performing a design activity (e.g., [Pahl and Beitz, 1984; French, 1985; Koller, 1985; Takeda, Veerkamp, Tomiyama and Yoshikawa, 1990; Brumsen, Pannekeet and Treur, 1992; Smithers, Corne and Ross, 1994; Klein, 2000]).

- *Design (control) strategy*, or *design scenario*
  A plan of design activities that governs the steps of a design process, such as trial-and-error, hierarchical decomposition, or pattern matching (e.g., [Akin, 1978; Mostow, 1985; Brown and Chandrasekaran, 1989; Goel and Pirolli, 1989; Treur, 1989; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Takeda, Veerkamp, Tomiyama and Yoshikawa, 1990; Strelnikov and Dmitrevich, 1991; Brumsen, Pannekeet and Treur, 1992; Logan and Smithers, 1992; Smithers, Corne and Ross, 1994; Klein, 2000; Varejão, De Menezes, Garcia, De Souza and Fromherz, 2000]).

- *Design conflict*
  Anything that stands in the way of reaching a design solution (e.g., [Alexander, 1964; Alexander and Poyner, 1970; Koller, 1985; Tong, 1987; Smithers, Corne and Ross, 1994; Klein, 2000] and all papers from *AIEDAM*, Special Issue on Conflict Management in Design, Vol. 9, No. 4, 1995).

- *Design decision*
  A decision made during a design process about the next design step to take (e.g., [Asimow, 1962; Pahl and Beitz, 1984; Koller, 1985; Mostow, 1985; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Brumsen, Pannekeet and Treur, 1992; Klein, 2000]).

- *Design trace*
  A sequence of design states within a design process (e.g., [Logan and Smithers, 1992; Smithers, Corne and Ross, 1994; Klein, 2000]).

- *Design pattern*
  A recurring sequence within design traces of design processes (e.g., [Asimow, 1962; Pahl and Beitz, 1984; French, 1985; Goel and Pirolli, 1989; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Gero, 1998]).

- *Design rationale*
  A set of justifications of the design decisions within a design trace or design pattern (e.g., [Mostow, 1985; Gruber and Russel, 1990; Klein, 1992; McKerlie and McLean, 1994; Smithers, Corne and Ross, 1994; Smithers, 1998; Burge and Brown, 2000]).

- *Design history*
  A set of one or more design traces of design processes (in the past), together with their design rationale (e.g., [Logan and Smithers, 1992; Chung and Goodwin, 1994; Smithers, Corne and Ross, 1994; Charlton and Wallace, 2000]).

- *Creativity in design*
  The act of creating design knowledge, that is, discovering, evaluating, and adapting concepts during a design process (e.g., [Boden, 1990; Logan and Smithers, 1992; Zhao and Maher, 1992, Candy and Edmonds, 1996; Fischer and Nakakoji, 1997; Aihara and Hori, 1998; Gero, 1998]).

- *Learning in design*
  The act of acquiring new design knowledge during a design process (e.g., [Fischer and Nakakoji, 1997; Gero, 1998; Sim and Duffy, 2000]).

- *Situatedness in design*
  The act of learning the situation-dependent applicability conditions of specific design knowledge (e.g., [Smithers, 1998; Reffat and Gero, 2000]).

- *Distributed design*
  The involvement of multiple (co-operative) design agents in a design process, each with their own, situated design knowledge (e.g., [Grecu and Brown, 1996; Campbell, Cagan and Kotovsky, 2000]).

2.2 Design Methods

A *design method* is a normative description of how to represent design problems and solutions as well as how to design in a specific application domain (e.g., [Pahl and Beitz, 1984; French, 1985; Maher, 1990; Bernaras and Van de Velde, 1994; Smithers, 1996]). The norms on which a design method is based usually come down to a few design principles. A *design principle* is an invariant for a design process, such as to maintain the independence of requirements or to minimise the amount of information included by the design solution (e.g., [Asimow, 1962; Pahl and Beitz, 1984; French, 1985; Goel and Pirolli, 1989; Pugh, 1990; Suh, 1990]).

According to most design researchers, a design method should contain the following representations:

- *Design problem space representation*
  A representation of a space of design problems (e.g., [Tong, 1987; Brown and Chandrasekaran, 1989; Chandrasekaran, 1990; Zhao and Maher, 1992; Bernaras and Van de Velde, 1994; Löckenhoff and Messer, 1994; Runkel, Balkany and Birmingham, 1994; Wielinga and Schreiber, 1997]).

- *Design solution space representation*
  A representation of a space of design solutions (e.g., [Tong, 1987; Brown and Chandrasekaran, 1989; Chandrasekaran, 1990; Zhao and Maher, 1992; Bernaras and Van de Velde, 1994; Runkel, Balkany and Birmingham, 1994; Löckenhoff and Messer, 1994; Wielinga and Schreiber, 1997]).

- *Design knowledge representation*
  A representation of process control knowledge (i.e., control flow in a design process), search control knowledge (e.g., propose-and-revise and constraint satisfaction) and application domain knowledge (e.g., [Tong, 1987; Goel and Chandrasekaran, 1989; Brown and Chandrasekaran, 1989; Chandrasekaran, 1990; Zhao and Maher, 1992; Bernaras and Van de Velde, 1994; Löckenhoff and Messer, 1994; Runkel, Balkany and Birmingham, 1994; Wielinga and Schreiber, 1997]).

These three types of representations may be combined into a *design prototype*, which organises knowledge about a set of similar design situations and which forms a framework for studying design processes (e.g., [Koller, 1985; Tong, 1987; Coyne, Rosenman, Radford, Balachandran and Gero, 1990; Gero, 1990; Zhao and Maher, 1992]).

2.3 Design Support Systems

In this thesis, the term *design support systems* refers to software systems, which provide to varying degrees suitable, helpful, co-operative and non-prescriptive support for a human designer trying to solve a design problem [Smithers, Corne and Ross, 1994], ranging from intelligent design assistants to fully automated design systems. A design support system should provide the following functionality (in terms of components of the system):

- *User interfacing*
  A facility for communication between a designer and a design support system, supporting the way that the designer likes to work (e.g., [Hammond, Davenport and Fitzpatrick, 1993; Bradley, Agogino and Wood, 1994; Candy and Edmonds, 1994]).

- *Requirements engineering*
  A facility for aiding the acquisition and modelling of design requirements (e.g., [Sumi, Hori and Ohsuga, 1998]).

- *Consistency maintenance*
  A facility for maintaining consistency between multiple design goals (e.g., [Logan and Smithers, 1992; Petrie, Cutkosky and Park, 1994]).

- *Context management*
  A facility for maintaining alternatives or solutions to particular design sub-problems and the selection of the most promising candidates for further development (e.g., [Logan and Smithers, 1992; Petrie, Cutkosky and Park, 1994]).

- *Knowledge representation*
  A facility for representing knowledge about the application domain and about the design process (e.g., [Logan and Smithers, 1992; Sun and Faltings, 1994; Ursu and Hammond, 2000]).

- *Inference*
  A facility for assigning values to design parameters and evaluating the resulting partial design solutions (e.g., [Logan and Smithers, 1992; Takeda and Nishida, 1994]).

- *Criticising*
  A facility for criticising partial design solutions on the basis of the design goals, the intentions for which these solutions have been developed or on the basis of regulatory design knowledge such as safety regulations or aesthetics (e.g., [Fischer, Nakakoji, Ostwald, Stahl and Sumner, 1993; Nakakoji, Sumner and Harstad, 1994; Stolze, 1994; Ursu and Hammond, 2000]).

2.4 Implications for AI in Design Research

This section describes the design topics (in terms of design concepts, design method representations and design support system functionality) that have to be investigated in more depth in order to further a better understanding of design processes and the development of useful design support systems.

Design theories and models. The main challenge is to account for the explorative nature of design (i.e., the phenomenon that in a design process, design problems are developed together with their solutions). Some theories and models of design ignore this issue altogether, by assuming a well defined design problem at the outset of a design process. Although this may be a reasonable assumption in some domains, for design processes in general it is not.

To this end, there is still much fundamental research to be done. This thesis provides a theory of design processes, which defines as many of the prevalent design concepts brought forth by design research as possible.

Design methods. In the course of time, many design methods have been developed for different application domains, with suitable (and usually domain-specific) representations of design problem space, design solution space and design knowledge. These methods are usually based on years of practice, and hardly ever on theories of design. This makes it difficult to explain and compare design methods, to transfer design methods from one application domain to another and to assess the suitability (e.g., pros and cons) of a specific design method for a specific application domain.

The main challenge therefore is to found design methods on a theory of design. Having such a theory enables a developer of design methods to design, analyse, compare and evaluate design methods more rigorously. Most AI in Design researchers agree that, to this end, a knowledge-level theory of design is an important prerequisite. Therefore, the theory of design processes provided by this thesis is formulated as a knowledge-level theory, using logic as a means. (Hence, in this thesis it is referred to as a logical theory of design.)

Design support systems. The same arguments put forward for design methods also apply to design support systems. If such systems are well-founded, it is easier for a developer to design, analyse, compare and evaluate design support systems. Furthermore, a knowledge-level theory of design offers a good basis for standardising design support system functionality such as consistency maintenance, context management, knowledge representation, inference, critiquing and (to a certain extent) user interfacing.

When developing a design support system, the computational aspects of design processes have to be considered. To make the development of computationally feasible functionality easier, this thesis offers a generic model of design. Based on our logical theory of design, this model describes at the knowledge level the essential types of information and knowledge that play a role within a design process, irrespective of design method and application domain.

Chapter 3

Static Aspects of Design

A logical theory of design is a theory that characterises design processes at a logical level. The logical theory of design in this thesis addresses not only the static aspects *of design (i.e., aspects of individual 'snapshots' of a design process) but also the* dynamic aspects *of design (i.e., aspects of possible sequences of 'snapshots' of a design process). This chapter focuses on the static aspects, and the next chapter on the dynamic aspects.*

Publications. *The development of a logical theory of design was part of the same research project that resulted in the generic design model described in Chapters 6 to 8. An initial version of the logical theory has been presented at the AID '94 Conference in Lausanne, Switzerland [Brazier, Langen, Ruttkay and Treur, 1994], and a refined, improved version at the IFIP WG5.2 Workshop in Mexico City, Mexico [Brazier, Langen and Treur, 1995*a*; Brazier, Langen and Treur, 1996].*

Chapters 3 and 4 present a logical, knowledge-level theory of design that focuses on the anatomy of design. This theory applies to any design process, irrespective of the domain of application, procedural aspects, and knowledge representation. This chapter starts by explaining the intended use of the theory.

Firstly, the theory furthers understanding of design processes. It acts as a frame of reference for the development, capture, dissemination, and use of knowledge about design processes. The theory offers concepts and logical concept relationships (i.e., an ontology), which can be taught, studied, questioned, tested, and applied for specific purposes. Furthermore, the theory is intended as a basis for further research, either to make a better theory or to develop a more elaborate theory that applies to a specific domain of application, for instance.

Secondly, the theory supports the development of design support systems. It describes the services a design support system must provide and (to some extent) how these services must be organised. Furthermore, the theory can be used for verification purposes, in order to identify strengths and weaknesses of the system. For this purpose, use can be made of the results of research on the verification of knowledge-based systems (e.g., [Treur and Willems, 1994; Leemans, Treur and Willems, 2002; Cornelissen, Jonker and Treur, 2002]).

As a knowledge-level theory, it does not restrict the way in which these services are implemented: the theory tells the system developer *what* to represent, not *how* to represent it. Chapter 1 explains that logic is an appropriate tool for analysing design processes at the knowledge level, because it is declarative and does not enforce a particular representation ([Newell, 1982]). Logic has already proven its use in many disciplines. In the field of diagnosis, for example, Reiter as well as Console and Torasso have developed logical theories that are acknowledged to be valuable contributions, although neglecting the dynamics of diagnostic processes [Reiter, 1987; Console and Torasso, 1990]. Furthermore, a large advantage of developing a logical theory is that a *semantics* can be defined that clearly defines the meaning of the syntactical constructs of the language.

Our logical theory of design captures two different types of aspects of design processes: static aspects, and dynamic aspects. Defining the static aspects and the dynamic aspects of a design process at the knowledge level results in an overview of concepts and logical concept relationships.

The *static aspects*, described in this chapter, concern the possible descriptions involved in a design process, such as sets of design requirements, or design object descriptions. The logical relationships between these types of descriptions include, for example, consistency of design object descriptions, and satisfaction of a design requirement by a design object description.

The *dynamic aspects*, described in the next chapter, concern the possible states and the sequences of state transitions within a design process, such as modification of a set of design requirements, or deductive refinement of a design object description. The logical relationships include, for example, the notion of a design solution.

This chapter is organised as follows. Section 3.1 introduces the concepts and their logical relationships to be defined. Section 3.2 introduces *order-sorted predicate logic*, a knowledge-level language to express the static aspects of design. The semantics of this type of logic accounts for the incompleteness of information in an individual state of a design process about the design object, its interactions with other objects, and the design requirements imposed on the design object, which is inevitable in most practical cases. Section 3.3 defines *partial models* for a formal interpretation of the static aspects of design processes.

Section 3.4 introduces the notion of *design object description*, which captures the whole of the information about a design object and its environment in a specific state of a design process. The section also defines generic relations between design object descriptions, such as deductive refinement.

Section 3.5 introduces *requirement qualification sets*: sets of design requirements on the design object that are to be satisfied by design object descriptions. The section also defines generic relations between requirement qualification sets, such as consistency and refinement.

Section 3.6 introduces generic relations between design object descriptions and individual design requirements, such as satisfaction, and also generic relations between design object descriptions and requirement qualification sets, such as fulfilment. Section 3.7 briefly discusses this logical theory of design in relation to the work of others with respect to the static aspects of design.

3.1 Conceptual Analysis of the Static Aspects of Design

Extending the list of concepts introduced in Chapter 2, Table 3.1 enumerates the concepts that constitute the static aspects of design, and explains the relationships between these concepts.

Table 3.1. Concepts that constitute the static aspects of design.

Concept	*Explanation*
Object	Something visible or tangible. Also: domain object.
Object attribute	A qualitative or quantitative characteristic of an individual object.
Object property	A specific object's attribute and its value.
Object relationship	A relationship between objects.
Object composition	A *part-of* relationship between an object and other objects (its parts).
Domain object information	A declaration of one or more properties or relationships of objects.
Domain object knowledge	The whole of domain object information that applies to every possible situation in a specific application domain.
Context of an object	Domain object information concerning the objects with which the given object is deemed to interact.
Behaviour of an object	Domain object information concerning interactions of the given object with other objects.
Design object description state (DOD state)	The whole of the (partial) domain object information concerning a design object, as available in a specific situation during a design process.
DOD state space	The space of possible design object description states for a specific application domain.
Design object description (DOD)	An intended design object description state, describing the structure or form of a specific object and a prescription for its construction.
Function of an object	The intended behaviour (i.e., use or purpose) of the given object.
Need and/or desire	An often implicit motivation of a human to change the individual or societal situation.
Design requirement	An explicit formulation of (part of) a need or desire.

Concept	*Explanation*
Requirement	A design requirement on the function, behaviour, or structure of an object to be constructed. Also: functional specification.
Qualified requirement	A design requirement formulated as an expression of the priority of satisfying one or more specific requirements. Also: design criterion.
Design requirement information	Information about the design requirements concerning a design object, in terms of their definitions and which ones are actually imposed.
Design requirement knowledge	The whole of design requirement information that applies to every possible set of design requirements in a specific application domain.
Requirement qualification set state (RQS state)	Situation-specific design requirement information concerning a specific design object. Also: requirements description, or design program.
RQS state space	The space of possible requirement qualification set states for a specific application domain.
Requirement qualification set (RQS)	An intended requirement qualification set state, describing the design requirement information applicable to a specific design object.
Basic assessment	Information about the satisfaction relation between one or more design object descriptions and one or more design requirements.
DOD satisfaction assessment	A basic assessment, concerning information about whether a specific design object description satisfies or violates specific design requirements.
Design requirement satisfiability assessment	A basic assessment, concerning information about whether a specific design requirement is satisfiable (by some design object description) or not.
DOD assessment	An assessment of a design object description, concerning information about whether this description fulfils or fails to fulfil specific requirement qualification sets, or epistemic information concerning basic assessments involving this description.
RQS assessment	An assessment of a requirement qualification set, concerning information about whether this set can be fulfilled (by some design object description), or epistemic information concerning basic assessments involving this set.

To be able to provide formal definitions of the concepts in Table 3.1, a logical language is needed of which the semantics accounts for the incompleteness of information in an individual state of a design process about the design object, its interactions with other objects, and the design requirements imposed on the design object. Such a logical language is introduced in the next section.

3.2 Order-Sorted Predicate Logic

A language with a clear semantics is needed to express the static aspects of design processes. As explained earlier in Section 1.3 of Chapter 1, logic is deemed appropriate for analysing

the knowledge level, since it is declarative and does not enforce a particular representation. In this section, the language of *order-sorted predicate logic* is introduced. This type of logic extends classical (first-order) predicate logic with sorts, to express knowledge such as 'A bicycle is a kind of vehicle.' Order-sorted predicate logic is just as powerful as classical predicate logic, but has a notational advantage.

In the language of first-order predicate logic, sentences are strings of characters arranged to follow precise rules of grammar. It includes *logical operators* such as $\neg$ (not), $\wedge$ (and), $\vee$ (or) and $\Rightarrow$ (implies), which are used to form complex sentences from simple ones. It further includes *quantifiers* $\forall$ (for all) and $\exists$ (there exists) which refer to all or some of the elements of the universe of discourse, respectively, without enumerating them. *Variable symbols* (or, *variables*) refer to individual elements of the universe of discourse without identifying them. *Object symbols* (also called *constants*) name specific elements of the universe of discourse, *function symbols* designate functions on elements, and *relation symbols* (also called *predicates*) name relations between elements.

In an order-sorted first-order language, *sort symbols* name specific sets of elements of the universe of discourse (e.g., the set of all colours), just as unary relations do. Sorts do not extend the expressive power of the language—using sorts just makes writing and reading sentences a bit easier. For instance, in an unsorted first-order language, the knowledge that all bright colours are vivid could be expressed as

$$(\forall x)\ (((\text{Colour}(x) \wedge \text{Bright}(x)) \Rightarrow \text{Vivid}(x)),$$

whereas in an order-sorted language, it could be expressed as

$$(\forall x\text{: COLOUR})\ (\text{Bright}(x) \Rightarrow \text{Vivid}(x)).$$

Using an order-sorted language, nouns in natural language representations of knowledge (such as "colours") can be distinguished from adjectives (such as "bright" and "vivid"), as nouns are mapped to sorts and adjectives to relations. Compared to classical predicate logic, this feature provides an additional means to structure the available domain knowledge.

The following definitions of signature, well-formed terms and well-formed formulas are based on those provided by Hedtstück and Schmitt, with some slight terminological changes [Hedtstück and Schmitt, 1989].

Definition 3.2.1. (Signature) An *order-sorted signature* $\Sigma = (\mathbf{S}, \mathbf{C}, \mathbf{F}, \mathbf{P})$ consists of

1. a partially ordered set of *sorts* $(\mathbf{S}, \leq)$ with least element $\bot$ (*bottom*) and greatest element $\top$ (*top*). The pair $(\mathbf{S}, \leq)$ is called the *sort hierarchy* and does not contain infinitely increasing chains.

2. an **S**-indexed family of sets of *object symbols* (or, *constants*) $(\mathbf{C}_S)_{S \in \mathbf{S}}$. A constant $c \in \mathbf{C}_S$ is written c: S.

3. an $\mathbf{S}^* \times \mathbf{S}$-indexed family of sets of *function symbols* $(\mathbf{F}_{w, S})_{w \in \mathbf{S}^*, S \in \mathbf{S}}$. A function $f \in \mathbf{F}_{w, S}$ is written f: $S_1 \ldots S_n \rightarrow S$, where $w = S_1 \ldots S_n$ is the *arity* and S the *co-arity* of f. The elements of w are called the *argument sorts* of f.

4. an $\mathbf{S}^*$-indexed family of sets of *predicate symbols* $(\mathbf{P}_w)_{w \in \mathbf{S}^*}$. A predicate $p \in \mathbf{P}_w$ is written p: $S_1 \ldots S_n$ where $w = S_1 \ldots S_n$.

The sets of symbols for sorts, objects, functions, and predicates are assumed to be pairwise disjoint. The least element $\perp$ is assumed not to occur in the arities and co-arities of **C**, **F**, and **P**. The predicate symbol = is reserved for identity propositions and is assumed to be included in **P**. Finally, in order to avoid problems with empty sorts, every sort except for $\perp$ is assumed to have at least one ground term (i.e., a term in which no variable occurs).

Example 3.2.2. A simple order-sorted signature for a universe of discourse about colours is:

({⊥, COLOUR, PRIMARY-COLOUR, SECONDARY-COLOUR, ⊤},
{Blue, Red, Yellow: PRIMARY-COLOUR, Green, Orange, Purple: SECONDARY-COLOUR},
{Opposite: COLOUR → COLOUR},
{Bright, Vivid: COLOUR, '=': COLOUR × COLOUR}),

with PRIMARY-COLOUR ≤ COLOUR and SECONDARY-COLOUR ≤ COLOUR.

Definition 3.2.3. (Well-Formed Terms) Given an order-sorted signature $\Sigma = (\mathbf{S}, \mathbf{C}, \mathbf{F}, \mathbf{P})$, a family of *sorted variables* over Σ is an **S**-indexed family of variables $V = \{V_S \mid S \in \mathbf{S}\}$ where $V_\perp = \varnothing$. The family of *well-formed terms* over Σ, TERMS(Σ), is the least **S**-indexed family of sets such that

1. For every $x \in V_S$, $x \in$ TERMS(Σ); the sort of x is S.
2. For every $c \in \mathbf{C}_S$, $c \in$ TERMS(Σ); the sort of c is S.
3. If $t_1, \ldots, t_n$ are terms in TERMS(Σ) of sort $S'_1, \ldots, S'_n$, respectively, then $f(t_1, \ldots, t_n) \in$ TERMS(Σ) if f: $S_1 \times \ldots \times S_n \rightarrow S$ and $S'_i \leq S_i$ for $i \in \{1, \ldots, n\}$; the sort of $f(t_1, \ldots, t_n)$ is S.

A *well-formed ground term* is a well-formed term in which no variable occurs.

Example 3.2.4. Given the sorted variable c: COLOUR, possible well-formed terms over the signature of Example 3.2.2 are Red, Opposite(Yellow), and Opposite(Opposite(c: COLOUR)).

Definition 3.2.5. (Well-Formed Formulas) Let $\Sigma = (\mathbf{S}, \mathbf{C}, \mathbf{F}, \mathbf{P})$ be an order-sorted signature, and V a family of sorted variables over Σ. The set of *well-formed formulas* over Σ, $\mathrm{WFF}(\Sigma)$, is the least set such that:

1. Let $t_1, \ldots, t_n$ be well-formed terms of sorts $S'_1, \ldots, S'_n$, respectively, and let $p: S_1 \ldots S_n$ with $S'_i \leq S_i$ for $i \in \{1, \ldots, n\}$. Then $p(t_1, \ldots, t_n)$ is a well-formed formula.

2. If φ is a well-formed formula, so is $(\neg\varphi)$.

3. If φ and ϕ are well-formed formulas, so are $(\varphi \wedge \phi)$, $(\varphi \vee \phi)$, $(\varphi \Rightarrow \phi)$ and $(\varphi \Leftrightarrow \phi)$.

4. If x is a sorted variable of sort S and φ is a well-formed formula, so are $(\forall x: S)(\varphi)$ and $(\exists x: S)(\varphi)$.

Example 3.2.6. Given the sorted variable x: COLOUR, possible well-formed formulas over the order-sorted signature of Example 3.2.2 are Opposite(Orange) = Blue, Bright(Yellow), and $(\forall x: \mathrm{COLOUR})$ (Opposite(Opposite(x)) = x).

Brackets are often omitted in formulas. In the sequel, the following notations and conventions are used.

- The symbol Σ (with or without subscript) denotes an order-sorted signature.
- The unary functions S, C, F, and P can be used to refer to the components of an order-sorted signature. For $\Sigma = (\mathbf{S}, \mathbf{C}, \mathbf{F}, \mathbf{P})$, $\mathrm{S}(\Sigma) = \mathbf{S}$, $\mathrm{C}(\Sigma) = \mathbf{C}$, $\mathrm{F}(\Sigma) = \mathbf{F}$, and $\mathrm{P}(\Sigma) = \mathbf{P}$.
- A *sentence* is a well-formed formula without free variables.
- The set $\mathrm{AT}(\Sigma) \subset \mathrm{WFF}(\Sigma)$ is the set of all *ground atoms* over Σ (i.e. atomic sentences including only well-formed ground terms).
- The set $\mathrm{LIT}(\Sigma) = \mathrm{AT}(\Sigma) \cup \{\neg\varphi \mid \varphi \in \mathrm{AT}(\Sigma)\}$ is the set of all *ground literals* over Σ.
- Infix notation is used for the identity relation, such as in f(A, B) = g(C).

3.3 Partial Models

During a design process, a designer defines a set of design requirements that reflects the needs and desires of a customer, and makes a satisfiable description of the design object to be constructed. The designer may start 'from scratch' with an initial set of design requirements and a blank design object description, which are then subject to a series of modifications. In practice, it is virtually inevitable that during a design process, the designer has to continually deal with incomplete information about the design requirements, the design object, and its environment. To capture the semantics of the static aspects of design, it is therefore important to account for the incompleteness of such information.

For this purpose, logic offers a suitable technique. *Partial models* make it possible to express the incompleteness of the available information in different states of a design process. In the examples in Section 3.2 concerning a universe of discourse about colours, for instance, a partial model may be used to express that the colour yellow is known to be bright, but not known to be vivid as well, and that the opposite of the colour purple is undefined.

Partial model theory is based on *three-valued logic*, introduced among others by Kleene [Kleene, 1952]. In this type of logic, it is no longer assumed that sentences can always be classified to be either true or false. Partial models extend classical models by their property that sentences can be interpreted to be neither true nor false, but undefined. For more details about partial model theory, see for instance the work of Blamey and Langholm [Blamey, 1986; Langholm, 1988]. Van Langen and Treur have shown that partial models are well suited to formalise *information states*: possibly incomplete descriptions of the different situations that may occur in a universe of discourse [Langen and Treur, 1989].

The definition of partial semantics is based on the strong Kleene semantics for the interpretation of well-formed propositional formulas composed of the logical connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ ([Kleene, 1952]). This approach follows the work on partial models by Blamey and Langholm [Blamey, 1986; Langholm, 1988]. Figure 3.1 shows the corresponding truth tables.

φ	$\neg\varphi$
1	0
0	1
u	u

$\varphi \wedge \phi$	1	0	u
1	1	0	u
0	0	0	0
u	u	0	u

$\varphi \vee \phi$	1	0	u
1	1	1	1
0	1	0	u
u	1	u	u

$\varphi \Rightarrow \phi$	1	0	u
1	1	0	u
0	1	1	1
u	1	u	u

$\varphi \Leftrightarrow \phi$	1	0	u
1	1	0	u
0	0	1	u
u	u	u	u

FIGURE 3.1. Kleene's strong three-valued connectives.

In this section, Definitions 3.3.1 to 3.3.6 have been restricted purposefully to the partial semantics of proposition logic, which is easier to read and understand. Readers familiar with order-sorted predicate logic and interested in their partial semantics may skip Definitions 3.3.1 to 3.3.6 and refer to Appendix A.

Definition 3.3.1. (Proposition-Logical Signature) A *proposition-logical signature* Σ is a set of proposition symbols (or, propositions).

Propositions are usually denoted by letters from the alphabet (most often *p* and *q*), with or without subscripts.

Definition 3.3.2. (Proposition-Logical Sentences) Given a proposition-logical signature Σ, the set of *(proposition-logical) sentences* over Σ, WFF(Σ), is the least set such that:

1. If $p \in \Sigma$, then p is a sentence.
2. If φ is a sentence, so is $(\neg\varphi)$.
3. If φ and ϕ are sentences, so are $(\varphi \wedge \phi)$, $(\varphi \vee \phi)$, $(\varphi \Rightarrow \phi)$ and $(\varphi \Leftrightarrow \phi)$.

The partiality of the semantics of proposition-logical sentences appears as follows. For a partial proposition p, two different sets are used, p^+ and p^-; when p is undefined, its interpretation does not belong to either one of these sets. Such an interpretation is possible in a three-valued logic, which distinguishes *true* (denoted 1), *false* (denoted 0), and a third truth value, *undefined* (denoted u). The purpose of the non-classical truth value *undefined* is to indicate a state of partial ignorance about the truth value of a sentence.

Definition 3.3.3. (Partial Model) Given a proposition-logical signature Σ, a *partial Σ-model* M consists for each proposition $p \in \Sigma$ of at most one of the nullary relations p^+_M and p^-_M.

In addition, let $\mathbf{T} = \{0, 1, u\}$ be a set of *truth values*, where 0 denotes the truth value *false*, 1 denotes *true* and u denotes *undefined*. Then the partial semantics of terms and sentences is defined as follows.

Definition 3.3.4. (Partial Semantics) Let Σ be a proposition-logical signature and M a partial Σ-model. Then:

1. $M(p) = 1$ iff p^+_M;
   $M(p) = 0$ iff p^-_M;
   $M(p) = u$ otherwise (i.e., iff neither p^+_M nor p^-_M).

2. If $\varphi, \phi \in \mathrm{WFF}(\Sigma)$, then:
   $M(\neg\varphi) = 1$ iff $M(\varphi) = 0$,
   $M(\neg\varphi) = 0$ iff $M(\varphi) = 1$,
   $M(\neg\varphi) = u$ otherwise;

   $M(\varphi \wedge \phi) = 1$ iff $M(\varphi) = 1$ and $M(\phi) = 1$,
   $M(\varphi \wedge \phi) = 0$ iff $M(\varphi) = 0$ or $M(\phi) = 0$,
   $M(\varphi \wedge \phi) = u$ otherwise;

   $M(\varphi \vee \phi) = 1$ iff $M(\varphi) = 1$ or $M(\phi) = 1$,
   $M(\varphi \vee \phi) = 0$ iff $M(\varphi) = 0$ and $M(\phi) = 0$,
   $M(\varphi \vee \phi) = u$ otherwise;

   $M(\varphi \Rightarrow \phi) = 1$ iff $M(\varphi) = 0$ or $M(\phi) = 1$,
   $M(\varphi \Rightarrow \phi) = 0$ iff $M(\varphi) = 1$ and $M(\phi) = 0$,
   $M(\varphi \Rightarrow \phi) = u$ otherwise;

$M(\varphi \Leftrightarrow \phi) = 1$ iff either $M(\varphi) = 1$ and $M(\phi) = 1$, or $M(\varphi) = 0$ and $M(\phi) = 0$,
$M(\varphi \Leftrightarrow \phi) = 0$ iff either $M(\varphi) = 1$ and $M(\phi) = 0$, or $M(\varphi) = 0$ and $M(\phi) = 1$,
$M(\varphi \Leftrightarrow \phi) = u$ otherwise.

Definition 3.3.5. (Truth and Completeness) Let Σ be a proposition-logical signature and M a partial Σ-model. A sentence $\varphi \in \text{WFF}(\Sigma)$ is *true* in M if $M(\varphi) = 1$, it is *false* in M if $M(\varphi) = 0$, and it is *undefined* in M if $M(\varphi) = u$. Furthermore, M is *complete* if every sentence is either true or false in M.

The notion of a complete partial model corresponds to the classical notion of a structure in standard logics.

Definition 3.3.6. (Satisfaction and Model) Let Σ be a proposition-logical signature and M a partial Σ-model. Then the *satisfaction* relation $\models$ is defined for all $\varphi \in \text{WFF}(\Sigma)$ as follows:

$M \models \varphi$ if $M(\varphi) = 1$; $M \not\models \varphi$ if $M(\varphi) = 0$.

If $M \models \varphi$, then M is a *model* of φ. For a subset $\Phi \subset \text{WFF}(\Sigma)$, $M \models \Phi$ denotes that M is a model of each element of Φ.

Sometimes it is useful to consider a restriction of a partial model to the interpretation of a subset of all possible sentences. In design, for example, a design object description includes information for the client and the designer (to be able to determine whether or not the given design requirements are satisfied) as well as for the constructor (to be able to construct the design object), and it may be useful to focus on the information for one of these parties. Also if different views on the design object description have been defined (e.g., the foundation, the outer walls, and the floors of an office building), it may be useful to consider different restrictions of the partial model. To this end, Definition 3.3.7 defines the notion of *reduct*.

Definition 3.3.7. (Reduct) Let Σ be a signature, $S \subseteq \text{AT}(\Sigma)$ a set of ground atoms, and M a partial Σ-model. Then $M \mid S$ denotes the *reduct* of M to S (or, an S-reduct of M), which is defined for ground atoms as follows:

$M \mid S(\varphi) = M(\varphi)$ for all $\varphi \in S$;
$M \mid S(\varphi) = u$ for all $\varphi \in \text{AT}(\Sigma) \setminus S$.

The satisfaction relation can be used to define the knowledge that a specific partial model has about some situation in a universe of discourse. This situation-specific knowledge is expressed as the set of sentences that are true in that partial model.

Definition 3.3.8. (Theory and Diagram of a Model) Let Σ be a signature and M a partial Σ-model. The *theory* of M, denoted Th(M), is the set $\{\varphi \in \text{WFF}(\Sigma) \mid M \text{ is a model of } \varphi\}$. The *diagram* of M, denoted Diag(M), is the set $\{\ \varphi \in \text{LIT}(\Sigma) \mid M \text{ is a model of } \varphi\ \}$.

A diagram (which is a subset of the theory of a model) is a useful notion to designate the factual information that a partial model has about a specific situation in the universe of discourse. The following relations between partial models are used to compare the information that two partial models have about the same situation in the universe of discourse.

Definition 3.3.9. (Refinement, Equivalence, and Inconsistency) Let Σ be a signature. A partial Σ-model M' is a *refinement* of a partial Σ-model M, denoted $M \leq M'$, if $\text{Th}(M) \subseteq \text{Th}(M')$. Furthermore, M and M' are *equivalent*, denoted $M \equiv M'$, if $M \leq M'$ and $M' \leq M$. If there is a sentence $\varphi \in \text{WFF}(\Sigma)$ such that $\varphi \in \text{Th}(M)$ and $(\neg\varphi) \in \text{Th}(M')$, then M and M' are *inconsistent* with each other, otherwise they are *consistent* with each other.

Figure 3.2 shows a practical application of partial models and refinements. Four drawings are shown, numbered 1 to 4. Drawing 1 shows a few basic parts of a van, such as the chassis and the wheels. Drawings 2 to 4 show more detailed but different versions of the same van. Partial models and the refinement relation between partial models can be used to express that drawings 2 to 4 each contain more information about the van than drawing 1.

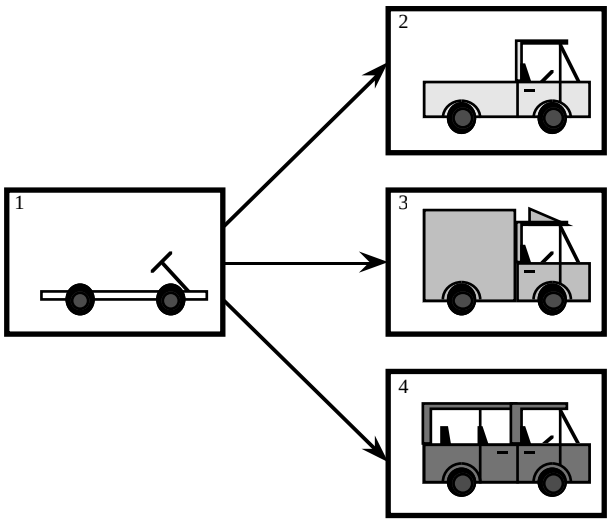


FIGURE 3.2. A drawing (left-hand side) and three more detailed versions (right-hand side).

Besides the information that a partial model has about some specific situation in a universe of discourse, there is also knowledge that applies to every situation in the universe of discourse. This knowledge is usually referred to as a *theory*.

Definition 3.3.10. (Theory) Let Σ be a signature. A Σ-*theory* $\Phi \subseteq \text{WFF}(\Sigma)$ is a consistent set of sentences (i.e., there exists at least one partial Σ-model M such that $M \models \Phi$).

A theory can be used to deduce implicitly present information from one partial model, which results in a new, more refined partial model that describes the same situation in the universe of discourse, but in more detail.

Definition 3.3.11. (Entailment and Consequence) Let Σ be a signature, M a partial Σ-model, and Φ a Σ-theory. Then a sentence $\varphi \in \text{WFF}(\Sigma)$ is *entailed* by M under Φ if for every complete partial Σ-model M' for which $M \leq M'$ and $M' \models \Phi$ hold, also $M' \models \varphi$ holds. The *consequence* of M under Φ, denoted $\text{DCons}(M, \Phi)$, is the set $\{\varphi \in \text{WFF}(\Sigma) \mid \varphi$ is entailed by M under $\Phi\}$.

Definition 3.3.12. (Deductive Refinement and Deductive Closure) Let Σ be a signature, and Φ a Σ-theory. Then a partial Σ-model M' is a *deductive refinement* of a partial Σ-model M under Φ, denoted $M \leq_\Phi M'$, if $M \leq M'$ and $\text{Th}(M') \subseteq \text{DCons}(M, \Phi)$. If $\text{Th}(M') = \text{DCons}(M, \Phi)$, then M' is called the *deductive closure* of M under Φ.

3.4 Design Object Descriptions

A basic concept in design is that of a *design object*: something that can (in principle) be seen, touched, or otherwise sensed. During a design process, a description of a design object is to be made, such that it contains sufficient information for the object to be manufactured, fabricated, constructed, or implemented in any other way. Design objects are typically thought to have a shape and physical appearance (e.g. buildings, machines, and appliances), but abstract things such as computer software also satisfy the definition of design object.

A design object is described through its *attributes*, of which each represents a qualitative or quantitative characteristic of the object, such as size, shape, or material. A *property* of an object is an attribute of the object and its value, such as that the colour of a given bicycle is red. Furthermore, a design object can be characterised through its *relationships* with other objects, such as that one system is compatible with another. A special type of object relationship is the *composition* of an object, which is essentially a part-of relationship between an object and other objects (its *parts*).

Domain object information is a declaration of one or more properties or relationships of objects. *Domain object knowledge* is the whole of domain object information that holds in every possible situation in a specific application domain (e.g., laws of physics). The *context*

or *environment* of an object is domain object information concerning the objects with which the object is deemed to interact. The *behaviour* of an object is domain object information concerning interactions of the object with other objects, such as the behaviour of a car on a slippery road.

To be able to formulate domain object information, a language is needed by means of which objects can be designated, as well as object attributes, properties, and relationships. Using order-sorted predicate logic as a basis, such a language is defined as follows.

Definition 3.4.1. (Domain Object Vocabulary) A *domain object vocabulary* is an order-sorted signature $\Sigma^{DO} = (\mathbf{S}, \mathbf{C}, \mathbf{F}, \mathbf{P})$ with the following properties:

- The sort DOMAIN-OBJECT $\in \mathbf{S}$ denotes domain objects in the universe of discourse.
- The sort ATTRIBUTE $\in \mathbf{S}$ denotes possible attributes of domain objects.
- The sort VALUE $\in \mathbf{S}$ denotes possible values of attributes. In the sort hierarchy $(\mathbf{S}, \leq)$, it holds that DOMAIN-OBJECT $\leq$ VALUE, denoting that one domain object may be the value of another domain object's attribute.
- The predicate HAS-VALUE: DOMAIN-OBJECT $\times$ ATTRIBUTE $\times$ VALUE denotes object properties, that is, the values assigned to specific attributes of specific domain objects.
- The predicate HAS-PART: DOMAIN-OBJECT $\times$ DOMAIN-OBJECT denotes object compositions, that is, the *part-of* relationship between specific domain objects.

Definition 3.4.2. (Domain Object Language) Given a domain object vocabulary Σ^{DO}, WFF(Σ^{DO}) is called the *domain object language* over Σ^{DO}. Each literal that is an element of LIT(Σ^{DO}) is called a *domain object information element*. The set BASIS $\subset$ WFF(Σ^{DO}) denotes the set of sentences that can be used to describe the basis structure of a design object.

Given Definitions 3.3.7 and 3.4.2, the BASIS-reduct of a partial Σ^{DO}-model M defines the information included in M that is needed for the construction of the concerned design object.

A *design object description information state*, or *design object description* for short, is the whole of the (partial) domain object information concerning a design object, as available in a specific situation of the design process. Different design object descriptions may apply to the same design object: one description may contain more information than the other about the design object (viz., one drawing being a later version than the other), or the descriptions (through reducts) may represent different views of the design object (viz., separate drawings of the infrastructures for water, sewage, and electricity in a building).

Definition 3.4.3. (Design Object Description (Information State)) Let Σ^{DO} be a domain object vocabulary. A *design object description (information state)* over Σ^{DO} is a partial Σ^{DO}-model. A design object description M is *empty* if Diag(M) = $\varnothing$.

To represent a design object description computationally, its diagram forms a good basis (see Definition 3.3.8). The diagram designates the factual information that a design object description includes. If more complex (i.e., non-atomic) information is of interest, then an inference engine can be used to derive this information from the available factual information in a design object description.

Given a domain object vocabulary, many different design object descriptions are possible (theoretically). Together, they constitute a space to be explored by a design process, which, in most practical applications, is vast.

Definition 3.4.4. (Design Object Description (Information State) Space) Let Σ^{DO} be a domain object vocabulary. The *design object description (information state) space*, $\text{IS}(\Sigma^{DO})$, is the set of all design object descriptions over Σ^{DO}.

A structure that can be imposed on a design object description state space is the binary *refinement relation*, which compares design object descriptions on the basis of the amount of domain object information they include. Such a comparison can be made independent of the way the design object description state space is explored during a design process (which is a dynamic aspect of design). The refinement relation holds between two design object description states if one includes all the domain object information that the other does.

Definition 3.4.5. (Design Object Description Refinement) Let Σ^{DO} be a domain object vocabulary. The *design object description refinement relation* is a refinement relation between pairs of design object descriptions over Σ^{DO}.

For a specific application domain, not every theoretically possible design object description in the design object description state space models a design that is practically possible, for instance because it includes different elements of domain object information that are inconsistent with each other. In contrast, design object descriptions that model designs that are practically possible are consistent with the *domain object knowledge*, that is, the whole of domain object information that applies to every possible situation in the application domain. Its formalisation is called a *domain object theory.*

Definition 3.4.6. (Domain Object Theory) Let Σ^{DO} be a domain object vocabulary. A *domain object theory* over Σ^{DO} is a Σ^{DO}-theory.

Representation of a domain object theory on a computer is easy if its sentences are formulated as *literal rules*: sentences of the form $\forall x_1 \ldots \forall x_K ((p_1 \wedge \ldots p_L) \Rightarrow (q_1 \wedge \ldots q_M))$, where $p_1, \ldots, p_L$ and $q_1, \ldots, q_M$ are literals in which zero or more variables from $\{x_1, \ldots, x_K\}$ occur. Using literal rules as a basis does not essentially restrict the expressive power of a domain object theory.

3.5 Requirement Qualification Sets

The *function* of an object is the intended behaviour (i.e., use or purpose) of the object. The primary trigger for design is that someone needs or desires something that is able to perform a specific function, but that does not exist in the current situation. A *need* or a *desire* is an often implicit motivation of a human to change the individual or societal situation, whereas a *design requirement* is an explicit formulation of (a part of) a need or desire.

Lawson distinguishes four different generators of design requirements [Lawson, 1997]: the designer (or, the design team), the client (i.e., the party paying for the design), the user (i.e., the party using the design object), and the legislator (i.e., the party monitoring laws and regulations). These generators are ordered on diminishing flexibility of the design requirements they put forward: while the designer's design requirements are usually flexible and optional, those of the legislator are rigid and mandatory (e.g., fire safety regulations).

In line with Lawson's analysis, among design requirements a distinction can be made between requirements and qualified requirements. A *requirement* is a statement of the function of a design object. Such a statement may range from abstract (e.g., "The bridge should stun everybody who sees it") to concrete (e.g., "The area of the kitchen should be between 30 m^2 and 35 m^2"). Preferably, the statement is formulated in such a way that it is possible to determine whether the requirement is satisfied or violated by some description of the design object.

It is usually useful to know how important a specific requirement is, compared to other requirements. One reason is that it may not possible to satisfy all stated requirements, due to inconsistencies, lack of time (because of a deadline), and so forth. In such cases, a designer is forced to make choices between alternative sets of requirements that can be imposed on the design object. A *qualified requirement* helps to make such choices: it is a statement of the extent to which specific (combinations of) requirements are preferred, either in absolute terms or in relation to other specific (combinations of) requirements.

For example, common qualifications used by designers are *hard* and *soft*. A requirement that is *hard* must be satisfied by a description of the design object. A requirement that is soft need not be satisfied by a description of the design object; however, all other things being equal, a design object description that satisfies this soft requirement is preferred to a design object description that does not.

Design requirement information is a declaration of a design requirement or a relationship on design requirements. *Design requirement knowledge* is the whole of design requirement information that holds in every possible situation in a specific application domain (e.g., fire safety regulations).

To be able to formulate design requirement information and knowledge, a language is needed to name design requirements, and to specify their definitions and the relationships between design requirements. This design requirement language is partly based on the domain object language. Reasoning about the design object's attributes and its relations with other domain objects is *object-level reasoning*, and forms the base of reference for all rea-

soning about the universe of discourse in design. Reasoning about requirements of the design object, as well as other aspects such as assumptions on the design object or its environment, is *meta-level reasoning*, as it is reasoning *about* object-level reasoning. For readers interested in an integrated approach to computational multi-level reasoning, see, for instance, the work of Weyhrauch and Maes [Weyhrauch, 1980; Maes, 1987; Maes, 1988]. A design requirement language based on order-sorted predicate logic is defined as follows.

Definition 3.5.1. (Design Requirement Vocabulary) Given a domain object vocabulary $\Sigma^{DO} = (\mathbf{S}^{DO}, \mathbf{C}^{DO}, \mathbf{F}^{DO}, \mathbf{P}^{DO})$, a *design requirement vocabulary* for Σ^{DO} is an order-sorted signature $\Sigma^{RQ} = (\mathbf{S}^{RQ}, \mathbf{C}^{RQ}, \mathbf{F}^{RQ}, \mathbf{P}^{RQ})$ with the following properties (see Tables 3.2 to 3.5):

Table 3.2. Standard sorts within $\mathbf{S}^{RQ}$.

Sort	*Explanation*
DOMAIN-OBJECT-INFO-ELEMENT	Atoms over the domain object vocabulary Σ^{DO}.
DOMAIN-OBJECT-INFO-FORMULA	Well-formed formulas over the domain object vocabulary Σ^{DO}. The relation DOMAIN-OBJECT-INFO-ELEMENT ≤ DOMAIN-OBJECT-INFO-FORMULA holds, which means that every atom over Σ^{DO} is a formula over Σ^{DO}.
REQUIREMENT-NAME	Possible names of requirements (for ease of reference).
REQUIREMENT	Possible requirements. The relations REQUIREMENT-NAME ≤ REQUIREMENT and DOMAIN-OBJECT-INFO-FORMULA ≤ REQUIREMENT both hold.
QUALIFIED-REQUIREMENT-NAME	Possible names of qualified requirements (for ease of reference).
INTEGER	Integers (to be used in the formulation of numeric constraints).
NUMERIC-CONSTRAINT	Possible numeric constraints (to be used in the formulation of qualifications of requirements).
QUALIFICATION	Possible qualifications of requirements.
REQUIREMENT-LIST	Possible lists of requirements (to be used in the formulation of qualified requirements). The relation REQUIREMENT ≤ REQUIREMENT-LIST holds.
QUALIFIED-REQUIREMENT-EXPRESSION	Possible expressions of qualified requirements.
QUALIFIED-REQUIREMENT	Possible qualified requirements. The relations QUALIFIED-REQUIREMENT-NAME ≤ QUALIFIED-REQUIREMENT and QUALIFIED-REQUIREMENT-EXPRESSION ≤ QUALIFIED-REQUIREMENT both hold.
DESIGN-REQUIREMENT	Possible design requirements. The relations REQUIREMENT ≤ DESIGN-REQUIREMENT and QUALIFIED-REQUIREMENT ≤ DESIGN-REQUIREMENT both hold.

Table 3.3. Standard constants within $\mathbf{C}^{RQ}$.

Constant	*Explanation*
‘Zero’, ‘0’: INTEGER	The integer zero.
‘Any’: NUMERIC-CONSTRAINT	Constraint denoting any positive number, but one is enough.
‘AllPossible’: NUMERIC-CONSTRAINT	Constraint denoting any positive number, as high as possible.
‘Every’: QUALIFICATION	Qualification denoting that every item from a list counts.
‘Nil’, ‘[]’: REQUIREMENT-LIST	The empty list of requirements.

Table 3.4. Standard functions within $\mathbf{F}^{RQ}$.

Function	*Explanation*
‘Succ’, ‘+1’: INTEGER → INTEGER	The successor of an integer.
‘AtLeast’: INTEGER → NUMERIC-CONSTRAINT	Constraint denoting “at least *the given number*.”
‘Exactly’: INTEGER → NUMERIC-CONSTRAINT	Constraint denoting “exactly *the given number*.”
‘AtMost’: INTEGER → NUMERIC-CONSTRAINT	Constraint denoting “at most *the given number*.”
‘AtRandom’: NUMERIC-CONSTRAINT → QUALIFICATION	Qualification denoting that the order of items in a list that is subject to the given constraint is insignificant.
‘InPreferredOrder’: NUMERIC-CONSTRAINT → QUALIFICATION	Qualification denoting that the order of items in a list that is subject to the given constraint is significant.
‘Dot’: REQUIREMENT × REQUIREMENT-LIST → REQUIREMENT-LIST	The requirement list that has the given requirement as its head, and the given requirement list as its tail.
‘QualifiedRequirementExpr’: QUALIFICATION × REQUIREMENT-LIST → QUALIFIED-REQUIREMENT-EXPRESSION	The qualified requirement expression built from the given qualification and the given requirement list.
‘Not’: DOMAIN-OBJECT-INFO-FORMULA → DOMAIN-OBJECT-INFO-FORMULA	The negation of the given well-formed formula.
‘And’: DOMAIN-OBJECT-INFO-FORMULA × DOMAIN-OBJECT-INFO-FORMULA → DOMAIN-OBJECT-INFO-FORMULA	The conjunction of the two given well-formed formulas.
‘Or’: DOMAIN-OBJECT-INFO-FORMULA × DOMAIN-OBJECT-INFO-FORMULA → DOMAIN-OBJECT-INFO-FORMULA	The disjunction of the two given well-formed formulas.
‘Implies’: DOMAIN-OBJECT-INFO-FORMULA × DOMAIN-OBJECT-INFO-FORMULA → DOMAIN-OBJECT-INFO-FORMULA	The implication of the two given well-formed formulas.
‘Exists’: S × DOMAIN-OBJECT-INFO-FORMULA → DOMAIN-OBJECT-INFO-FORMULA	The existentially quantified well-formed formula, built from the given variable from sort $S \in \mathbf{S}^{DO}$ and the given well-formed formula.
‘ForAll’: S × DOMAIN-OBJECT-INFO-FORMULA → DOMAIN-OBJECT-INFO-FORMULA	The universally quantified well-formed formula, built from the given variable from sort $S \in \mathbf{S}^{DO}$ and the given well-formed formula.

A design requirement that is presented at the start of a design process, or that is generated during the design process, need not be imposed on the design object at every stage of the design process. For example, the design requirement may be initially considered, then temporarily neglected, and reconsidered at a later time. The predicate IS-TO-BE-SATISFIED can be used to point out the individual requirements and qualified requirements that must be satisfied in the current stage of the design process.

Table 3.5. Standard predicates within $\mathbf{P}^{RQ}$.

Predicate	*Explanation*
IS-DEFINED-AS: REQUIREMENT-NAME× DOMAIN-OBJECT-INFO-FORMULA	Statement about the well-formed formula that corresponds to the given requirement name.
IS-DEFINED-AS: QUALIFIED-REQUIREMENT-NAME× QUALIFIED-REQUIREMENT-EXPRESSION	Statement about the qualified requirement expression that corresponds to the given qualified requirement name.
IS-TO-BE-SATISFIED: DESIGN-REQUIREMENT	Statement about a design requirement that must be satisfied.

Definition 3.5.2. (Design Requirement Language) Given a design requirement vocabulary Σ^{RQ}, WFF(Σ^{RQ}) is called the *design requirement language* over Σ^{RQ}. Each literal that is an element of LIT(Σ^{RQ}) is called a *design requirement information element*.

During design, not all design requirements of a design object are necessarily considered at the same time. For example, at the start of designing, only a number of the design requirements may be available. Furthermore, when a designer uses different views to distinguish different aspects of a design object (e.g., the hull, the electric wiring, and the piping of a ship), usually only a number of the available design requirements of the design object are considered for each individual view of the design object. As a result, at any point in time the description of the design requirements of a design object may be partial.

Definition 3.5.3. (Requirement Qualification Set (Information State)) Let Σ^{RQ} be a design requirement vocabulary. A *requirement qualification set (information state)* over Σ^{RQ} is a partial Σ^{RQ}-model. A requirement qualification set M is *empty* if Diag(M) = ∅.

To represent a requirement qualification set computationally, its diagram forms a good basis. The same arguments given for the representation of design object descriptions apply to the representation of requirement qualification sets.

Given a design requirement vocabulary, many different requirement qualification sets are possible (theoretically). Together, they constitute a space to be explored by a design process, which, in most practical applications, is vast.

Definition 3.5.4. (Requirement Qualification Set (Information State) Space) Let Σ^{RQ} be a design requirement vocabulary. The *requirement qualification set (information state) space*, $IS(\Sigma^{RQ})$, is the set of all requirement qualification sets over Σ^{RQ}.

A structure that can be imposed on a requirement qualification set state space is the binary *refinement relation*, which compares requirement qualification sets on the basis of the amount of design requirement information they include. Such a comparison can be made independent of the way the requirement qualification set state space is explored during a design process (which is a dynamic aspect of design). The refinement relation holds between two requirement qualification sets if one includes all the design requirement information that the other does.

Definition 3.5.5. (Requirement Qualification Set Refinement) Let Σ^{RQ} be a design requirement vocabulary. The *requirement qualification set refinement relation* is a refinement relation between pairs of requirement qualification sets over Σ^{RQ}.

For a specific application domain, not every requirement qualification set in the requirement qualification set state space models a plausible set of design requirements. Specific sets of design requirements may be impossible due to, for example, inconsistency. In contrast, requirement qualification sets that model sets of design requirements that are practically possible are consistent with the *design requirement knowledge*, that is, the whole of design requirement information that applies to every possible situation in the application domain. Its formalisation is called a *design requirement theory.*

Definition 3.5.6. (Design Requirement Theory) Let Σ^{RQ} be a design requirement vocabulary. A *design requirement theory* over Σ^{RQ} is a Σ^{RQ}-theory.

As for a domain object theory, a design requirement theory can be represented computationally in a straightforward manner if its sentences are formulated as literal rules.

3.6 Generic Assessment Relations

An *assessment relation* is a relation between design object descriptions and requirement qualification sets or the design requirements that they include (as part of their design requirement information). There are two different types of assessment relations: a *basic assessment relation* is an assessment relation between design object descriptions and design requirements, and a *fulfilment relation* is an assessment relation between design object descriptions and requirement qualification sets.

Considering basic assessment relations first, one question is whether or not a specific design object description satisfies or violates a given requirement.

Definition 3.6.1. (Satisfaction of Requirements) Let Σ^{DO} be a domain object vocabulary, Φ a domain object theory over Σ^{DO}, D a design object description over Σ^{DO}, $F \in$ WFF(Σ^{DO}) a well-formed domain object formula, Σ^{RQ} a design requirement vocabulary, and S a requirement qualification set over Σ^{RQ}. For every requirement name R: REQUIREMENT-NAME, if S is a model of IS-DEFINED-AS(R, F), then:

- If D is a model of F, then D *satisfies* R. If the deductive closure of D under Φ satisfies R, then D satisfies R under Φ.
- If D is a model of ($\neg F$), then D *violates* R. If the deductive closure of D under Φ violates R, then D violates R under Φ.
- If either D satisfies R or D violates R (under Φ), then D is *decisive* with respect to R (under Φ).

Definition 3.6.1 can be naturally extended to the satisfaction of sets of requirements in the following manner:

- If D satisfies R_1, R_2, …, and R_N (i.e., every one), then D satisfies $\{R_1, R_2, \ldots R_N\}$.
- If D violates R_1, R_2, …, or R_N (i.e., at least one), then D violates $\{R_1, R_2, \ldots R_N\}$.
- If D either satisfies $\{R_1, R_2, \ldots R_N\}$ or violates $\{R_1, R_2, \ldots R_N\}$, then D is decisive with respect to $\{R_1, R_2, \ldots R_N\}$.

Another question, similar to the one above, is whether or not a specific design object description satisfies or violates a given qualified requirement.

Definition 3.6.2. (Satisfaction of Qualified Requirements) Let Σ^{DO} be a domain object vocabulary, Φ a domain object theory over Σ^{DO}, D a design object description over Σ^{DO}, Σ^{RQ} a design requirement vocabulary, S a requirement qualification set over Σ^{RQ}, Q a qualification, and L a list of requirements (which, for the purpose of this definition, may be considered a set). For every qualified requirement name QR: QUALIFIED-REQUIREMENT-NAME, if S is a model of IS-DEFINED-AS(QR, QualifiedRequirementExpr(Q, L)), then D *satisfies* QR if:

- Q = ‘Every’, and D satisfies L.
- Q = ‘AtRandom(Any)’ or Q = ‘InPreferredOrder(Any)’, and D satisfies at least one requirement $R \in L$.
- Q = ‘AtRandom(AllPossible)’ or Q = ‘InPreferredOrder(AllPossible)’.
- Q = ‘AtRandom(AtLeast(N))’ or Q = ‘InPreferredOrder(AtLeast(N))’, and the number of requirements from L that are satisfied by D is greater than or equal to N.
- Q = ‘AtRandom(Exactly(N))’ or Q = ‘InPreferredOrder(Exactly(N))’, and the number of requirements from L that are satisfied by D is equal to N.
- Q = ‘AtRandom(AtMost(N))’, L)) or Q = ‘InPreferredOrder(AtMost(N))’, and the number of requirements from L that are satisfied by D is less than or equal to N.

If S is a model of IS-DEFINED-AS(QR, QualifiedRequirementExpr(Q, L)), then D *violates* QR if:

- Q = 'Every', and D violates L.
- Q = 'AtRandom(Any)' or Q = 'InPreferredOrder(Any)', and D violates every requirement $R \in L$.
- Q = 'AtRandom(AtLeast(N))' or Q = 'InPreferredOrder(AtLeast(N))', and the number of requirements from L that are violated by D is greater than $|L| - N$.
- Q = 'AtRandom(Exactly(N))' or Q = 'InPreferredOrder(Exactly(N))', and the number of requirements from L that are violated by D is greater than $|L| - N$, or the number of requirements from L that are satisfied by D is greater than $|L| - N$.
- Q = 'AtRandom(AtMost(N))' or Q = 'InPreferredOrder(AtMost(N))', and the number of requirements from L that are satisfied by D is greater than $|L| - N$.

If the deductive closure of D under Φ satisfies QR, then D satisfies QR under Φ, and if the deductive closure of D under Φ violates QR, then D violates QR under Φ. Finally, if either D satisfies QR or D violates QR (under Φ), then D is *decisive* with respect to QR (under Φ).

Definition 3.6.2 can be naturally extended to the satisfaction of sets of qualified requirements in the following manner:

- If D satisfies QR_1, QR_2, …, and QR_N, then D satisfies $\{QR_1, QR_2, \ldots QR_N\}$.
- If D violates QR_1, QR_2, …, or QR_N, then D violates $\{QR_1, QR_2, \ldots QR_N\}$.
- If D either satisfies $\{QR_1, QR_2, \ldots QR_N\}$ or violates $\{QR_1, QR_2, \ldots QR_N\}$, then D is decisive with respect to $\{QR_1, QR_2, \ldots QR_N\}$.

Based on the above basic assessment relations, the next definition formulates a number of assessment properties of design requirements (i.e., qualified requirements or requirements).

Definition 3.6.3. (Design Requirement Assessment) Let Σ^{DO} be a domain object vocabulary, and DR_1, DR_2, …, DR_N one or more names of design requirements. Then:

- If there exists a design object description over Σ^{DO} that satisfies $\{DR_1, DR_2, \ldots DR_N\}$, then $\{DR_1, DR_2, \ldots DR_N\}$ is *satisfiable*.
- If every complete design object description over Σ^{DO} satisfies $\{DR_1, DR_2, \ldots DR_N\}$, then $\{DR_1, DR_2, \ldots DR_N\}$ is *tautological*.
- If every complete design object description over Σ^{DO} violates $\{DR_1, DR_2, \ldots DR_N\}$, then $\{DR_1, DR_2, \ldots DR_N\}$ is *contradictory*.
- If there does not exist a BASIS-reduct of a design object description over Σ^{DO} that is decisive with respect to $\{DR_1, DR_2, \ldots DR_N\}$, then $\{DR_1, DR_2, \ldots DR_N\}$ is *imprecise*.

For fulfilment relations, similar questions as the ones raised above are whether or not a specific design object description fulfils or fails to fulfil a given requirement qualification set.

Definition 3.6.4. (Fulfilment) Let Σ^{DO} be a domain object vocabulary, Φ a domain object theory over Σ^{DO}, Σ^{RQ} a design requirement vocabulary, *D* a design object description over Σ^{DO}, and *S* a requirement qualification set over Σ^{RQ}. Then:

- If *D* satisfies each design requirement *DR* for which it holds that *S* is a model of IS-TO-BE-SATISFIED(*DR*), then *D fulfils S*. If the deductive closure of *D* under Φ fulfils *S*, then *D* fulfils *S* under Φ.
- If *D* violates at least one design requirement *DR* for which it holds that *S* is a model of IS-TO-BE-SATISFIED(*DR*), then *D fails to fulfil S*. If the deductive closure of *D* under Φ fails to fulfil *S*, then *D* fails to fulfil *S* under Φ.
- If *D* either fulfils *S* or fails to fulfil *S* (under Φ), then *D* is *decisive* with respect to *S* (under Φ).

For a specific requirement qualification set, relevant questions are whether or not the set can be fulfilled, and whether or not it is *ill structured* ([Simon, 1973]). Smithers, Corne, and Ross interpret Simon's definition of ill-structuredness as follows [Smithers, Corne and Ross, 1994]. Using their terminology, the initial requirements description derived from the motivating statement of need and/or desire may be:

- *incomplete* in the sense that it fails to identify some necessary criteria that must be met if the needs and/or desires are to be satisfied;
- *inconsistent* in the sense that it may specify two or more mutually incompatible criteria—incompatible in principle or in practice;
- *imprecise* in the sense that it fails to specify sufficiently refined criteria (to be able to judge whether or not all given criteria are satisfied by a given design description);
- *ambiguous* in the sense that it fails to identify an ordering or prioritisation on the importance of the given criteria.

Equating a requirements description (in the terminology of Smithers, Corne, and Ross) with a requirement qualification set (in our terminology), the consequence of being ill structured is the following: a requirement qualification set that exhibits one or more of the above properties may have to be modified during design in order to make it possible to generate a satisfactory design object description.

Definition 3.6.5. (Requirement Qualification Set Assessment) Let Σ^{DO} be a domain object vocabulary, Σ^{RQ} a design requirement vocabulary, Φ a design requirements theory over Σ^{RQ}, and *S* a requirement qualification set over Σ^{RQ}. Then:

- If there exists a design object description over Σ^{DO} that fulfils *S*, then *S can be fulfilled*.
- If *S* is not equivalent to its deductive closure under Φ, then *S* is *incomplete*.
- If every complete design object description over Σ^{DO} fails to fulfil *S*, then *S* is *inconsistent*.
- If there does not exist a BASIS-reduct of a design object description that is decisive with respect to the fulfilment of *S*, then *S* is *imprecise*.
- If more than one complete design object description over Σ^{DO} fulfils *S*, then *S* is *ambiguous*.

Based on the definitions above, it is now possible to define a *design solution.* A more elaborate definition, including temporal aspects, is given in Chapter 4.

Definition 3.6.6. (Design Solution) Let Σ^{DO} be a domain object vocabulary, Φ a domain object theory over Σ^{DO}, and Σ^{RQ} a design requirements vocabulary. A design object description *D* over Σ^{DO} is a *design solution* (under Φ) to a requirement qualification set *S* over Σ^{RQ} if *D* is consistent and the BASIS-reduct of *D* fulfils *S* (under Φ).

3.7 Discussion

This chapter has presented the static aspects of design: concepts and their logical relationships that apply to each possible, individual state of a design process. For a knowledge-level analysis of these static aspects, first-order predicate logic has been used to express design object behaviour and design requirements. Furthermore, partial models in the form of design object descriptions and requirement qualification sets have been used as interpretations of design object behaviour and design requirements, respectively.

The definition of a design object description differs from what other researchers mean by a design description. For example, Coyne, Rosenman, Radford, Balachandran, and Gero use *design description* to denote the explicit information about a design, and *performance* to denote the set of things that can be declared about the design that is not explicitly stated, but that is implicit in the design description [Coyne, Rosenman, Radford, Balachandran, and Gero, 1990]. In our logical theory of design, a design object description may include both the explicit information contained in a design description, as well as the performance information. The similarity is that the BASIS-reduct of a design object description in our terms, which defines the information needed for the construction of a design object (see Definitions 3.3.7, 3.4.2, and 3.6.3), seems to correspond well to a design description in their terms.

One of the contributions of the work presented in this chapter is a formal definition of the notions of incompleteness, inconsistency, imprecision, and ambiguity provided by Smithers, Corne and Ross [Smithers, Corne and Ross, 1994]. This formal definition makes it possible to determine whether or not the initial requirement qualification set given to a design process is ill structured, which is a situation to be dealt with by many design processes in practice.

Chapter 4

Dynamic Aspects of Design

A logical theory of design is a theory that characterises design processes at a logical level. The logical theory of design in this thesis addresses not only the static aspects *of design (i.e., aspects of individual 'snapshots' of a design process) but also the* dynamic aspects *of design (i.e., aspects of possible sequences of 'snapshots' of a design process). The previous chapter focuses on the static aspects, and this chapter on the dynamic aspects.*

Publications. *The development of a logical theory of design was part of the same research project that resulted in the generic design model described in Chapters 6 to 8. An initial version of the logical theory has been presented at the AID '94 Conference in Lausanne, Switzerland [Brazier, Langen, Ruttkay and Treur, 1994], and a refined, improved version at the IFIP WG5.2 Workshop in Mexico City, Mexico [Brazier, Langen and Treur, 1995*a*; Brazier, Langen and Treur, 1996].*

Chapters 3 and 4 present a logical, knowledge-level theory of design that focuses on the anatomy of design. This theory applies to any design process, irrespective of the domain of application, procedural aspects, and knowledge representation.

This logical theory of design captures two different types of aspects of design processes: static aspects, and dynamic aspects. Defining the static aspects and the dynamic aspects of a design process at the knowledge level results in an overview of concepts and logical concept relationships.

The *static aspects*, described in Chapter 3, concern the possible descriptions involved in a design process, such as sets of design requirements, or design object descriptions. The logical relationships between these types of descriptions include, for example, consistency of design object descriptions, and satisfaction of a design requirement by a design object description.

The *dynamic aspects*, described in this chapter, concern the possible states and sequences of state transitions within a design process, such as modification of a set of design requirements, or deductive refinement of a design object description. The logical relationships that are part of the dynamic aspects include, for example, the notion of a design solution.

This chapter is organised as follows. Section 4.1 introduces the concepts and logical relationships to be defined. Section 4.2 introduces *temporal logic*, a knowledge-level language to express the dynamic aspects of design. The semantics of this type of logic accounts for the incompleteness of information in a design process about the design conflicts that may occur, the design decisions that can be made, and the effectiveness of design decisions, which is inevitable in most practical cases. Section 4.3 defines *partial temporal models* for a formal interpretation of the dynamic aspects of design processes.

Section 4.4 defines the notion of *design process state*, which captures the whole of the available information about a specific state of a design process. Section 4.5 introduces *design process steps* as transitions between design process states, and *design traces* as (possible) sequences of design process steps, each of which makes up for a specific design process. Section 4.6 briefly discusses this logical theory of design in relation to the work of others with respect to the dynamic aspects of design.

4.1 Conceptual Analysis of the Dynamic Aspects of Design

Extending the list introduced in Chapter 2, Table 4.1 enumerates the concepts that constitute the dynamic aspects of design, and explains the relationships between these concepts.

Table 4.1. Concepts that constitute the dynamic aspects of design.

Concept	*Explanation*
Design activity	An act of designing, such as the refinement or structuring of a design problem, and the generation, visual perception, or evaluation of a design solution. Also: design event, design (sub-)task, design operation, or design function.
Design process	A sequence of design activities, such as building design, industrial design, civil engineering, or software design.
Design (process) state	The design information known at a specific moment during a design process. Also: design stage.
Design step	A transition of a design state into a new design state as a result of performing a design activity. Also: design move, design change, or design revision.
Design (control) strategy	A plan of design activities that governs the steps of a design process, such as trial-and-error, hierarchical decomposition, or pattern matching. Also: design scenario.
Design conflict	Anything that stands in the way of reaching a design solution.
Design decision	A decision made during a design process about the next design step to take.
Design trace	A sequence of design states within a design process.

Concept	*Explanation*
Design pattern	A recurring sequence within design traces of design processes.
Design rationale	A set of justifications of the design decisions within a design trace or design pattern.
Design history	One or more design traces of design processes (in the past), together with their design rationale.
Creativity in design	The act of creating design knowledge, that is, discovering, evaluating, and adapting concepts during a design process.
Learning in design	The act of acquiring design knowledge during a design process.
Situatedness in design	The act of learning the situation-dependent applicability conditions of specific design knowledge.
Distributed design	The involvement of multiple (co-operative) design agents in a design process, each with their own situated design knowledge.

To be able to provide formal definitions of the concepts in Table 4.1, a logical language is needed of which the semantics accounts for the dynamic aspects of a design process, and in particular the incompleteness of information in a design process (e.g., about the effect of design decisions). Such a logical language is introduced in the next section.

4.2 Temporal Logic

In this thesis, a design process is viewed as a sequence of states in which decisions are taken about, for example, the strategy for the design process as a whole, about which modifications to apply to requirement qualification sets, and about which modifications to apply to design object descriptions. Each such decision depends in part on the past of the design process (e.g., which specific choices have been made earlier, and which alternatives have been identified). Furthermore, each such decision generally affects the future of the design process (e.g., which options are no longer available as a result of the choice currently made).

A language with a clear semantics is needed to express the dynamic aspects of design processes. As explained earlier in Section 1.3 of Chapter 1, logic is deemed appropriate for analysing the knowledge level, as it is declarative and does not enforce a particular representation. In this section, the language of *temporal logic* is introduced. This type of logic makes it possible to refer to states in a design process and the manifold relations between these states. Temporal logic extends classical first-order predicate logic with specific temporal predicates, to express knowledge such as ‘Currently, it has been decided to deductively refine the current design object description.’

The following definitions are based on those provided by Treur, with some slight terminological changes [Treur, 1994].

Definition 4.2.1. (Information State) Let Σ be an order-sorted signature. An *information state* for Σ is a partial Σ-model. The set of all information states for Σ is denoted $IS(\Sigma)$.

A sequence of information states can be generated by a series of transitions on information states. A first transition is applied to an information state, yielding a new information state. Then on this new state, a second transition is applied, and so on.

Definition 4.2.2. (Transition) Let Σ be an order-sorted signature. A Σ-*transition* is a pair of information states for Σ. A *transition relation* for Σ is a set of Σ-transitions, that is, a binary relation on $IS(\Sigma) \times IS(\Sigma)$. If a transition relation is defined as a mapping from $IS(\Sigma)$ into $IS(\Sigma)$, it is called a *transition function*.

Time points are used to distinguish different information states describing different situations that may occur in the course of a design process.

Definition 4.2.3. (Flow of Time) A *(discrete) flow of time* is a pair $\mathbf{T} = (T, \propto)$, where T is a non-empty set of time points, and $\propto$ is a binary relation on T. For $t_1, t_2 \in T$, $t_1 \propto t_2$ denotes that t_2 is the (immediate) *successor* of t_1. It is assumed that $\propto$ is irreflexive (i.e., for all $t \in T$, $t \propto t$ does not hold), antisymmetric (i.e., for all $t_1, t_2 \in T$, if $t_1 \propto t_2$ holds, then $t_2 \propto t_1$ does not hold), and antitransitive (i.e., for all $t_1, t_2, t_3 \in T$, if $t_1 \propto t_2$ and $t_2 \propto t_3$ hold, then $t_1 \propto t_3$ does not hold). The transitive closure of $\propto$ is written $<$ ('is less than'), which is assumed to be a linear ordering. **T** is *rooted* with root R if R is a unique least element in T (i.e., for all $t \in T$, either $R = t$ or $R < t$). Finally, **T** satisfies *successor existence* if every time point has a successor (i.e., for all $t_1 \in T$, there exists a $t_2 \in T$ such that $t_1 \propto t_2$).

In the sequel, it is assumed that flows of time are rooted, and that they satisfy successor existence. This is equivalent to **T** being isomorphic to the set of natural numbers. Therefore, in the remainder, the set of natural numbers will be used as the flow of time.

Definition 4.2.4. (Temporal Operator) A *temporal operator* is an element of the predicate set {**P**, **C**, **X**}, where

1. **P** refers to previous information states.
2. **C** refers to the current information state.
3. **X** refers to the next information state.

The temporal operators **P** ('In the past'), **C** ('Currently'), and **X** ('Next') denote when a given formula was, is, and will be true, respectively.

Definition 4.2.5. (Well-Formed Temporal Formulas) Let Σ be an order-sorted signature, and WFF(Σ) the set of well-formed formulas over Σ (cf. Definition 3.2.5 in Chapter 3). The set of *well-formed temporal formulas* over Σ, WFTF(Σ), is the least set such that:

1. For every sentence $\sigma \in$ WFF(Σ), $\mathbf{P}\sigma$, $\mathbf{C}\sigma$, and $\mathbf{X}\sigma$ are well-formed temporal formulas.
2. If φ and ϕ are well-formed temporal formulas, so are $(\neg\varphi)$, $(\varphi \wedge \phi)$, $(\varphi \vee \phi)$, $(\varphi \Rightarrow \phi)$, and $(\varphi \Leftrightarrow \phi)$.

Note that for the sake of simplicity, it is not possible to nest the temporal operators **P**, **C**, and **X**.

Example 4.2.6. "For the first time since the start (of the design process), the client commits to requirement qualification set RQS99 as an acceptable substitute for requirement qualification set RQS01."

$$\neg\mathbf{P}\text{IS-ACCEPTABLE-SUBSTITUTE-FOR}(\text{RQS99, RQS01}) \wedge$$
$$\mathbf{C}\text{IS-ACCEPTABLE-SUBSTITUTE-FOR}(\text{RQS99, RQS01})$$

4.3 Partial Temporal Models

During a design process, a designer defines a set of design requirements that reflects the needs and desires of a customer, and makes a satisfiable description of the design object to be constructed. Perhaps with the exception of routine design cases, it is inevitable that the designer does not have complete information about the course of the design process in advance. To capture the semantics of the dynamic aspects of design, it is therefore important to account for the incompleteness of such information.

In this thesis, an approach is followed that has been developed in the realm of temporal semantics (see [Engelfriet and Treur, 1994; Gavrila and Treur, 1994; Treur, 1994]). In this approach, the behaviour of a reasoning process is represented by a set of *reasoning traces*. Each trace expresses the reasoning steps taken sequentially in time, representing a particular line of reasoning that could be followed within the process, as shown in Figure 4.1. The traces A to C start with the same design object description in the states A_1 to C_1, respectively. Due to different design decisions, different design object descriptions are developed in these three traces. In trace C, the subsequent states C_2 and C_3 contain design object descriptions that are inconsistent with each other; perhaps the design object description in state C_2 has been modified to meet some violated requirements. In trace D, the subsequent states D_1 and D_2 contain design object descriptions that are also inconsistent with each other—perhaps the design object description in state D_1 has been replaced by the one in state D_2, and subsequently those two design object descriptions have been combined to form the one in state D_3.

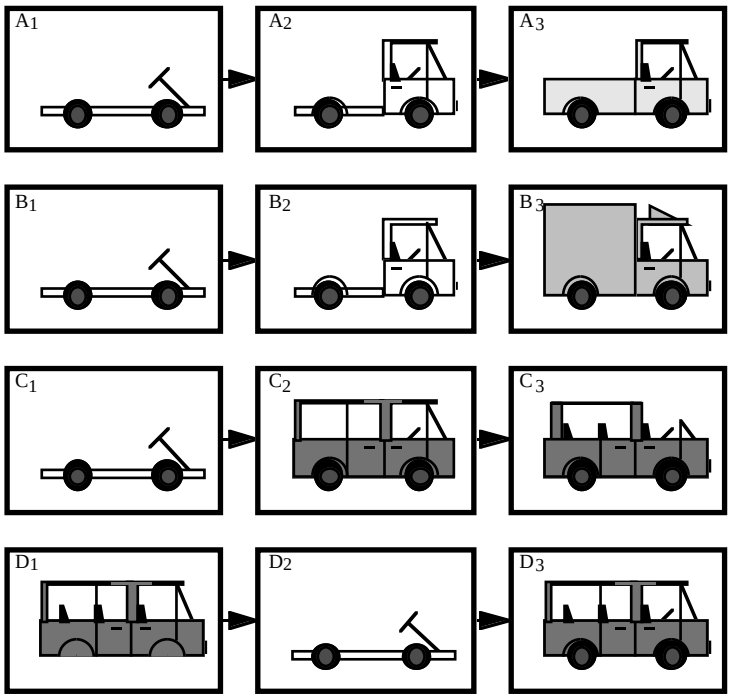


FIGURE 4.1. Four different partial temporal models showing successive design object descriptions.

A trace is formalised by a *partial temporal model*: a sequence of partial models, each of which formalises an information state resulting from a reasoning step. The advantage of this approach is that the semantics of the static aspects and the semantics of the dynamic aspects are structurally connected.

Definition 4.3.1. (Partial Temporal Model) Let Σ be a signature. A *partial temporal Σ-model*, or a *Σ-trace*, is a mapping M: $\mathbf{T} \rightarrow \mathrm{IS}(\Sigma)$. The set of all partial temporal Σ-models is denoted $\mathrm{TRACES}(\Sigma)$.

Definition 4.3.2. (Partial Temporal Semantics) Let Σ be a signature, M a partial temporal Σ-model, $t \in \mathbf{T}$ a time point, and $\models$ the satisfaction relation on $\mathrm{IS}(\Sigma) \times \mathrm{WFF}(\Sigma)$ (cf. Definition 3.3.6 in Chapter 3). Then the *temporal satisfaction* relations $\models^{+}_{\mathrm{T}}$ and $\models^{-}_{\mathrm{T}}$ are defined as follows:

1. If $\sigma \in \mathrm{WFF}(\Sigma)$ is a sentence, then:
   $M, t \models^{+}_{\mathrm{T}} \mathbf{P}\sigma$ iff there exists a time point $s \in \mathbf{T}$ such that $s < t$ and $M(s) \models \sigma$;
   $M, t \models^{-}_{\mathrm{T}} \mathbf{P}\sigma$ otherwise.

   $M, t \models^{+}_{\mathrm{T}} \mathbf{C}\sigma$ iff $M(t) \models \sigma$;
   $M, t \models^{-}_{\mathrm{T}} \mathbf{C}\sigma$ otherwise.

$M, t \models^{+}_{T} \mathbf{X}\sigma$ iff there exists a time point $u \in \mathbf{T}$ such that $t \propto u$ and $M(u) \models \sigma$;
$M, t \models^{-}_{T} \mathbf{X}\sigma$ otherwise.

2. If $\varphi, \phi \in$ WFTF(Σ), then:

$M, t \models^{+}_{T} \neg\varphi$ iff $M, t \models^{-}_{T} \varphi$;
$M, t \models^{-}_{T} \neg\varphi$ otherwise.

$M, t \models^{+}_{T} \varphi \wedge \phi$ iff $M, t \models^{+}_{T} \varphi$ and $M, t \models^{+}_{T} \phi$;
$M, t \models^{-}_{T} \varphi \wedge \phi$ otherwise.

$M, t \models^{+}_{T} \varphi \vee \phi$ iff $M, t \models^{+}_{T} \varphi$ or $M, t \models^{+}_{T} \phi$;
$M, t \models^{-}_{T} \varphi \vee \phi$ otherwise.

$M, t \models^{+}_{T} \varphi \Rightarrow \phi$ iff $M, t \models^{-}_{T} \varphi$ or $M, t \models^{+}_{T} \phi$;
$M, t \models^{-}_{T} \varphi \Rightarrow \phi$ otherwise.

$M, t \models^{+}_{T} \varphi \Leftrightarrow \phi$ iff $M, t \models^{+}_{T} \varphi$ and $M, t \models^{+}_{T} \phi$, or $M, t \models^{-}_{T} \varphi$ and $M, t \models^{-}_{T} \phi$;
$M, t \models^{-}_{T} \varphi \Leftrightarrow \phi$ otherwise.

The following notions are straightforward adaptations of the notions of truth, satisfaction, model, and theory of a model, as defined in Chapter 3.

Definition 4.3.3. (Temporal Truth) Let Σ be a signature and M a partial temporal Σ-model. A sentence $\varphi \in$ WFTF(Σ) is *true* in M at time t if $M, t \models^{+}_{T} \varphi$, and it is *false* in M at time t if $M, t \models^{-}_{T} \varphi$.

Definition 4.3.4. (Temporal Satisfaction) Let Σ be a signature, M a partial temporal Σ-model, and $t \in \mathbf{T}$ a time point. Then the *temporal satisfaction* relation $\models_{T}$ is defined for all $\varphi \in$ WFTF(Σ) as follows:

$M, t \models_{T} \varphi$ if $M, t \models^{+}_{T} \varphi$;
$M, t \not\models_{T} \varphi$ if $M, t \models^{-}_{T} \varphi$.

Definition 4.3.5. (Model) Let Σ be a signature and $\varphi \in$ WFTF(Σ) a well-formed temporal formula. A partial temporal Σ-model M is a *model* of φ, denoted $M \models_{T} \varphi$, if $M, t \models_{T} \varphi$ for every time point $t \in \mathbf{T}$. For a subset $\Phi \subset$ WFTF(Σ), $M \models_{T} \Phi$ denotes that M is a model of each element of Φ.

Definition 4.3.6. (Theory of a Model) Let Σ be a signature and M a partial temporal Σ-model. The *theory* of M, denoted Th(M), is the set $\{\varphi \in$ WFTF(Σ) | M is a model of $\varphi\}$.

4.4 Design Process States

In a design process, several activities can be distinguished. A *design activity* (or design event, design (sub-)task, design operation, or design function) is an act of designing, such as the refinement or structuring of a design problem, and the generation, visual perception, or evaluation of a design solution. A *design process* is a sequence of design activities, such as building design, industrial design, civil engineering, or software design.

A *design (process) state* is the design information known at a specific moment during a design process. This design information includes the current requirement qualification set and the current design object description being developed. It also includes information about suitable design activities, possible modifications to the current requirement qualification set, possible modifications to the current design object description, and the overall design strategy that governs the design process. In other words, design states contain design information that allows to reason at different *reflection levels* ([Maes, 1987; Maes, 1988]) about solving the design problem at hand, which, from bottom to top, are defined as follows:

- The *object level* includes information and knowledge about a specific application domain, in terms of the structure, form, and behaviour of objects that may occur in that domain. The knowledge at this level (e.g., physical laws) makes it possible to deduce properties of domain objects and relations between domain objects.
- The *first meta-level* includes information and knowledge about design requirements (stating what the design object is supposed to do and how it should behave) and design object descriptions (stating what the structure of the design object will be and how it will behave). The knowledge at this level makes it possible to derive implicit design requirements entailed by explicit design requirements, to determine goals for deriving implicit domain object information entailed by explicit domain object information, to determine defaults for missing domain object information, and to assess design object descriptions and design requirements with respect to each other.
- The *second meta-level* includes information and knowledge about requirement qualification sets, their relationships with design object descriptions, and design activities. The knowledge at this level makes it possible to determine goals for deriving implicit design requirement information, to determine modifications to requirement qualifications sets and design object descriptions, and to assess design object descriptions and requirement qualification sets with respect to each other.
- The *third meta-level* includes information and knowledge about design process objectives, overall design strategies, and design process evaluations. The knowledge at this level makes it possible to derive implicit design process objectives, and to determine an overall design strategy as well as more specific strategies for the manipulation of requirement qualification sets and design object descriptions, respectively.

Figure 4.2 shows nine types of design states, of which the following are specific to the dynamic aspects of design (indicated in the figure by means of rectangular boxes):

- At the first meta-level: *basic evaluation state*.
- At the second meta-level: *DOD alteration state* and *RQS alteration state.*
- At the third meta-level: *overall DOD control state*, *overall RQS control state*, *overall co-ordination state*, and—their combination—*overall design control state*.

Furthermore, Figure 4.2 shows five bi-directional reflection relations between the following pairs of design state types (explained in Section 4.5):

- design object description state (i.e., the *current design object description*) and basic evaluation state,
- basic evaluation state and DOD alteration state,
- requirement qualification set state (i.e., the *current requirement qualification set*) and RQS alteration state,
- DOD alteration state and overall DOD control state.
- RQS alteration state and overall RQS control state.

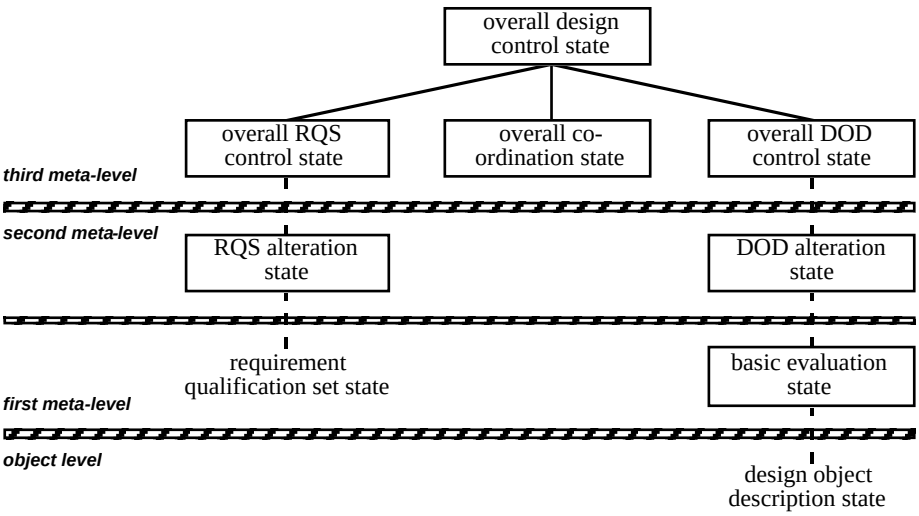


FIGURE 4.2. Design states, composition relations (solid lines) and reflection relations (dashed lines).

4.4.1 Basic Evaluation State

A *basic evaluation state* is a design process state at the first meta-level, linked through a reflection relation to a design object description state at the object level. It captures the following types of information that apply only to the current basis evaluation state:

- epistemic information about the domain object information included in the current design object description,
- default domain object information for the current design object description,
- goals to deduce domain object information from the current design object description,
- assertions and retractions of assumptions about domain object information for the current design object description,
- queries for design object descriptions.

In addition, a basic evaluation state captures the following types of information:

- epistemic information about the domain object information included in specific design object descriptions,
- assessments of design object descriptions in relation to design requirements, as well as assessments of (sets of) design requirements (as defined in Chapter 3).

Definition 4.4.1. (Basic Evaluation Vocabulary) Let $\Sigma^{DO} = (\mathbf{S}^{DO}, \mathbf{C}^{DO}, \mathbf{F}^{DO}, \mathbf{P}^{DO})$ be a domain object vocabulary, and $\Sigma^{RQ} = (\mathbf{S}^{RQ}, \mathbf{C}^{RQ}, \mathbf{F}^{RQ}, \mathbf{P}^{RQ})$ a design requirements vocabulary. A *basic evaluation vocabulary* based on $\langle \Sigma^{DO}, \Sigma^{RQ} \rangle$ is an order-sorted signature $\Sigma^{basic\text{-}eval} = (\mathbf{S}^{basic\text{-}eval}, \mathbf{C}^{basic\text{-}eval}, \mathbf{F}^{basic\text{-}eval}, \mathbf{P}^{basic\text{-}eval})$, where the sorts $\mathbf{S}^{basic\text{-}eval}$, constants $\mathbf{C}^{basic\text{-}eval}$, functions $\mathbf{F}^{basic\text{-}eval}$, and predicates $\mathbf{P}^{basic\text{-}eval}$ are defined as follows (see Tables 4.2 to 4.5):

Table 4.2. Standard sorts within $\mathbf{S}^{basic\text{-}eval}$.

Sort	*Explanation*
DOMAIN-OBJECT-INFO-ELEMENT	Well-formed atoms over the domain object vocabulary Σ^{DO}.
SIGN	Truth values in a three-valued logic.
DOD-NAME	Possible names of design object descriptions.
DESIGN-REQUIREMENT-LIST	Possible lists of design requirements. DESIGN-REQUIREMENT ≤ DESIGN-REQUIREMENT-LIST holds.

Table 4.3. Standard constants within $\mathbf{C}^{basic\text{-}eval}$.

Constant	*Explanation*
'Pos', 'Neg', 'Unk': SIGN	The truth values *true*, *false*, and *unknown/undefined*.
'Empty-DOD': DOD-NAME	Name denoting an empty design object description.
'Nil', '[]': DESIGN-REQUIREMENT-LIST	The empty list of design requirements.

Table 4.4. Standard functions within $\mathbf{F}^{basic\text{-}eval}$.

Function	*Explanation*
'Dot', '.': DESIGN-REQUIREMENT × DESIGN-REQUIREMENT-LIST → DESIGN-REQUIREMENT-LIST	A design requirement list that has the given design requirement as its head and the given design requirement list as its tail.

Table 4.5. Standard predicates within $\mathbf{P}^{\text{basic-eval}}$.

Predicate	*Explanation*
INCLUDES-DOMAIN-OBJECT-INFORMATION: DOD-NAME × DOMAIN-OBJECT-INFO-ELEMENT × SIGN	Statement about the domain object information included in the given design object description.
IS-CURRENTLY-INCLUDED: DOMAIN-OBJECT-INFO-ELEMENT × SIGN	Statement about the domain object information included in the current design object description.
HAS-DEFAULT-DOMAIN-OBJECT-INFORMATION: DOD-NAME × DOMAIN-OBJECT-INFO-ELEMENT × SIGN	Default domain object information determined for the given design object description.
IS-PART-OF-DEDUCTIVE-DOD-REFINEMENT-FOCUS: DOMAIN-OBJECT-INFO-ELEMENT × SIGN	Domain object information set as a goal for deductive refinement of the current design object description.
IS-TO-BE-ASSERTED: DOMAIN-OBJECT-INFO-ELEMENT × SIGN	Statement about domain object information to be added to the current design object description.
IS-TO-BE-RETRACTED: DOMAIN-OBJECT-INFO-ELEMENT × SIGN	Statement about domain object information to be deleted from the current design object description.
IS-INCLUDED-IN-WHICH-DODS: DOMAIN-OBJECT-INFO-ELEMENT × SIGN	Query for design object descriptions that include the given domain object information.
INCLUDES-WHICH-DOMAIN-OBJECT-INFORMATION: DOD-NAME	Query for the domain object information included in a given design object description.
SATISFIES: DOD-NAME× DESIGN-REQUIREMENT-LIST	Design requirements satisfied by the given design object description (cf. Definitions 3.6.1 and 3.6.2).
VIOLATES: DOD-NAME× DESIGN-REQUIREMENT-LIST	Design requirements violated by the given design object description (cf. Definitions 3.6.1 and 3.6.2).
CAN-BE-REFINED-TO-SATISFY: DOD-NAME× DESIGN-REQUIREMENT-LIST	Design requirements that can be satisfied by a refinement of the given design object description (cf. Definitions 3.6.1 and 3.6.2).
IS-DECISIVE-WRT-SATISFACTION-OF: DOD-NAME× DESIGN-REQUIREMENT-LIST	Design requirements satisfied or violated by any complete deductive refinement of the given design object description (cf. Definitions 3.6.1 and 3.6.2).
IS-SATISFIABLE: DESIGN-REQUIREMENT-LIST	Statement that the given set of design requirements is satisfiable (cf. Definition 3.6.3.). (Note that the design requirement list is used as a set.)
IS-TAUTOLOGICAL: DESIGN-REQUIREMENT-LIST	Statement that the given set of design requirements is tautological (cf. Definition 3.6.3.).
IS-CONTRADICTORY: DESIGN-REQUIREMENT-LIST	Statement that the given set of design requirements is contradictory (cf. Definition 3.6.3.).

Definition 4.4.2. (Basic Evaluation State and Space) Let $\Sigma^{\text{basic-eval}}$ be a basic evaluation vocabulary. A *basic evaluation state* over $\Sigma^{\text{basic-eval}}$ is a partial $\Sigma^{\text{basic-eval}}$-model. The *basic evaluation space* $\text{IS}(\Sigma^{\text{basic-eval}})$ is the set of all basic evaluation states over $\Sigma^{\text{basic-eval}}$.

4.4.2 Alteration States

DOD alteration states and RQS alteration states have a number of types of information in common. In this section, a generic alteration vocabulary is defined, which is first extended to a DOD alteration vocabulary and then to an RQS alteration vocabulary. To motivate the contents of an alteration vocabulary, publications of many (AI in Design) researchers can be cited. In the following, results of Lawson's research are used [Lawson, 1997].

In an attempt to define a route map of design processes, Lawson informally describes a design process as "a negotiation between a design problem and a design solution through the three activities of analysis, synthesis, and evaluation". That is, in his view a design process generally starts with some formulation of a design problem, and through analysis, synthesis, and evaluation (where the number of iterations and the order within an iteration is not fixed), it ends with a formulation of a design solution, if successful.

Lawson claims that it is often only during the design process itself that it becomes clear what the design problem is, and what a good design solution. He states that "the problem with design so often is that you cannot set sensible criteria for success unless you have some appreciation of what is possible."

Following Lawson's argument, this means that considerations or conclusions in a design process about what the design problem is, and what a design solution, are part of the dynamic aspects of design. In other words, an explanation for the formulations of the design problem and the design solution at the end of the design process must take into account the chain of steps that have been taken in the design process.

Studies examined or conducted by Lawson show that designers restrict the range of possibilities by initially focussing attention on a limited selection of design requirements and moving quickly towards some ideas about the solution. Lawson notes that good designers are able to sustain several parallel lines of thought, each representing a different idea, which may be eliminated as unworkable or unsatisfactory later in the design process, after which choices are made between the remaining ideas, possibly combining features of the alternative ideas.

Therefore, in a design process several states and state transitions must be distinguished, where each state captures one or many formulations of the design problem and one or many ideas about the design solution, as well as judgements of (the formulations of) the design problem and the design solution (i.e., which ones are rejected and which ones selected).

An *alteration vocabulary* is a vocabulary for expressions used in the analysis, synthesis, and evaluation of requirement qualification sets and design object descriptions. It defines the following types of information that apply only to the current alteration state:

- information about the identity of the current design object description,
- information about the identity of the current requirement qualification set,
- information about the process status (idle or active) of the design process,
- information about the past and future (expected) consumption of design resources (e.g., time and budget),
- information about design solutions (see Definitions 4.5.21 and 4.5.22 later on).

In addition, an alteration vocabulary defines the following types of information:

- epistemic information about the design requirement information included in specific requirement qualification sets,
- epistemic information about the basic evaluation information determined for specific requirement qualification sets,
- assessments of design object descriptions in relation to requirement qualification sets.

Definition 4.4.3. (Alteration Vocabulary) Let $\Sigma^{RQ} = (\mathbf{S}^{RQ}, \mathbf{C}^{RQ}, \mathbf{F}^{RQ}, \mathbf{P}^{RQ})$ be a design requirement vocabulary, and $\Sigma^{basic\text{-}eval} = (\mathbf{S}^{basic\text{-}eval}, \mathbf{C}^{basic\text{-}eval}, \mathbf{F}^{basic\text{-}eval}, \mathbf{P}^{basic\text{-}eval})$ a basic evaluation vocabulary. An *alteration vocabulary* based on $\langle \Sigma^{RQ}, \Sigma^{basic\text{-}eval} \rangle$ is an order-sorted signature $\Sigma^{alt} = (\mathbf{S}^{alt}, \mathbf{C}^{alt}, \mathbf{F}^{alt}, \mathbf{P}^{alt})$, where the sorts $\mathbf{S}^{alt}$, constants $\mathbf{C}^{alt}$, and predicates $\mathbf{P}^{alt}$ are defined as follows (see Tables 4.6 to 4.8):

Table 4.6. Standard sorts within $\mathbf{S}^{alt}$.

Sort	*Explanation*
DESIGN-REQUIREMENT-INFO-ELEMENT	Well-formed atoms over the design requirement vocabulary Σ^{RQ}.
BASIC-EVALUATION-INFO-ELEMENT	Well-formed atoms over the basic evaluation vocabulary $\Sigma^{basic\text{-}eval}$ (defined in Definition 4.4.1).
SIGN	Truth values in a three-valued logic.
RQS-NAME	Possible names of requirement qualification sets.
PROCESS-STATUS	Possible process statuses of an alteration process.
DESIGN-RESOURCE	Possible design resources used in an alteration process.

Table 4.7. Standard constants within $\mathbf{C}^{alt}$.

Constant	*Explanation*
'Pos', 'Neg', 'Unk': SIGN	The truth values *true*, *false*, and *unknown/undefined*.
'EmptyRQS': RQS-NAME	The empty requirement qualification set.
'Idle': PROCESS-STATUS	Process status *idle* (i.e., not engaged in reasoning).
'Active': PROCESS-STATUS	Process status *active* (i.e., engaged in reasoning).

Table 4.8. Standard predicates within $\mathbf{P}^{alt}$.

Predicate	*Explanation*
IS-CURRENT-DOD: DOD-NAME	Identity of the current design object description.
IS-CURRENT-RQS: RQS-NAME	Identity of the current requirement qualification set.
INCLUDES-DESIGN-REQUIREMENT-INFORMATION: RQS-NAME × DESIGN-REQUIREMENT-INFO-ELEMENT × SIGN	Statement about the design requirement information included in the given requirement qualification set.

Predicate	*Explanation*
INCLUDES-BASIC-EVALUATION-INFORMATION: RQS-NAME × BASIC-EVALUATION-INFO-ELEMENT × SIGN	Statement about basic evaluation information determined for the given requirement qualification set.
FULFILS: DOD-NAME× RQS-NAME	Statement that the given design object description fulfils the given requirement qualification set.
FAILS-TO-FULFIL: DOD-NAME× RQS-NAME	Statement that the given design object description fails to fulfil the given requirement qualification set.
CAN-BE-REFINED-TO-FULFIL: DOD-NAME× RQS-NAME	Statement that the given requirement qualification set can be fulfilled by a refinement of the given design object description.
IS-DECISIVE-WRT-FULFILMENT-OF: DOD-NAME× RQS-NAME	Statement that the given design object description is decisive with respect to the fulfilment of the given requirement qualification set.
IS-RQS-SOLUTION-TO: RQS-NAME × RQS-NAME	Statement that the first requirement qualification set is a solution to the second set.
IS-ACCEPTABLE-SUBSTITUTE-FOR: RQS-NAME × RQS-NAME	Statement that the client commits to the first requirement qualification set as an acceptable substitute for the second requirement qualification set.
IS-DOD-SOLUTION-TO: DOD-NAME× RQS-NAME	Statement that the given design object description is a solution to the given set.
IS-BASIS-REDUCT-OF: DOD-NAME× DOD-NAME	Statement that the first design object description is a reduct of the second description to the domain object information necessary for the construction of the design object.
IS-PREVIOUS-STATUS, IS-CURRENT-STATUS, IS-NEXT-STATUS: PROCESS-STATUS	Statement about the previous, current, and next status of a design process, respectively.
HAS-PAST-CONSUMPTION-OF: DESIGN-RESOURCE × NUMERIC-CONSTRAINT	Statement about the past consumption of a given design resource.
HAS-FUTURE-CONSUMPTION-OF: DESIGN-RESOURCE × NUMERIC-CONSTRAINT	Statement about the expected consumption of a given design resource.

A *DOD alteration state* is a design process state at the second meta-level, linked through a reflection relation to a basic evaluation state at the first meta-level. It not only captures information as defined by the alteration vocabulary, but also the following types of information that apply only to the current DOD alteration state:

- information about which alterations are proposed, selected or rejected for application to the current design object description,

- information about which design object description (to be retrieved from the history of the design process) is to become the current design object description,
- information about the basis (i.e., input) for and the results (i.e., output) of assessing design object descriptions against design requirements, determining a focus for the deductive refinement of the current design object description, determining default domain object information for specific design object descriptions, and formulating queries for design object descriptions.

In addition, a DOD alteration state captures information about the composition of design object description alterations. For the sake of simplicity, it is assumed that each alteration is a set of modifications.

It has already been remarked in Chapter 3 that to computationally represent a design object description, its diagram forms a good basis. Given that the diagram is the set of all domain object information literals of which the design object description is a model, it suffices for creating an alteration to a design object description to use constructs for adding and deleting domain object information literals.

Definition 4.4.4. (DOD Alteration Vocabulary) Let $\Sigma^{alt} = (\mathbf{S}^{alt}, \mathbf{C}^{alt}, \mathbf{F}^{alt}, \mathbf{P}^{alt})$ be an alteration vocabulary. A *DOD alteration vocabulary* based on Σ^{alt} is an order-sorted signature $\Sigma^{DOD\text{-}alt} = (\mathbf{S}^{DOD\text{-}alt}, \mathbf{C}^{DOD\text{-}alt}, \mathbf{F}^{DOD\text{-}alt}, \mathbf{P}^{DOD\text{-}alt})$, which extends Σ^{alt}, and where the sorts $\mathbf{S}^{DOD\text{-}alt}$, functions $\mathbf{F}^{DOD\text{-}alt}$, and predicates $\mathbf{P}^{DOD\text{-}alt}$ are defined as follows (see Tables 4.9 to 4.11):

Table 4.9. Standard sorts within $\mathbf{S}^{DOD\text{-}alt}$.

Sort	*Explanation*
DOMAIN-OBJECT-INFO-LITERAL	Possible domain object information literals.
DOD-MODIFICATION	Elementary modifications of design object descriptions.
DOD-ALTERATION	Alterations (i.e., sets of modifications) of design object descriptions. DOD-MODIFICATION ≤ DOD-ALTERATION holds.
DOD-SPECIFIC-BASIS-INFO-ELEMENT, DOD-SPECIFIC-RESULTS-INFO-ELEMENT	Well-formed atoms over the basic evaluation vocabulary $\Sigma^{basic\text{-}eval}$ (where the two sorts consist of the same objects, but are used for different purposes).

Table 4.10. Standard functions within $\mathbf{F}^{DOD\text{-}alt}$.

Function	*Explanation*
DOMAIN-OBJECT-INFORMATION: DOMAIN-OBJECT-INFO-ELEMENT × SIGN → DOMAIN-OBJECT-INFO-LITERAL	A domain object information literal formed by a given domain object information element and a given truth value sign.
ADDITION-OF, DELETION-OF: DOMAIN-OBJECT-INFO-LITERAL → DOD-MODIFICATION	Modification to the current design object description by adding/deleting a given domain object information literal.

Table 4.11. Standard predicates within $\mathbf{P}^{\text{DOD-alt}}$.

Predicate	*Explanation*
INCLUDES-DOD-MODIFICATION: DOD-ALTERATION × DOD-MODIFICATION	Statement that the given design object description alteration includes the given modification.
IS-PROPOSED-DOD-ALTERATION: DOD-ALTERATION	A proposed alteration to the current description.
IS-REJECTED-DOD-ALTERATION: DOD-ALTERATION	A rejected alteration to the current description.
IS-SELECTED-DOD-ALTERATION: DOD-ALTERATION	A selected alteration to the current description.
IS-REPLACEMENT-FOR-CURRENT-DOD: DOD-NAME	Statement that the given design object description is to become the current design object description.
IS-PART-OF-DOD-MODIFICATION-BASIS: DOD-SPECIFIC-BASIS-INFO-ELEMENT × SIGN	Statement about the information to be assumed as a basis for the assessment of design object descriptions against design requirements, etc.
IS-PART-OF-DOD-MODIFICATION-RESULTS: DOD-SPECIFIC-RESULTS-INFO-ELEMENT × SIGN	Statement about the epistemic information about the results of the assessment of design object descriptions against design requirements, etc.

Definition 4.4.5. (DOD Alteration State, Theory, and Space) Let $\Sigma^{\text{DOD-alt}}$ be a DOD alteration vocabulary. A *DOD alteration state* over $\Sigma^{\text{DOD-alt}}$ is a partial $\Sigma^{\text{DOD-alt}}$-model. A *DOD alteration theory* over $\Sigma^{\text{DOD-alt}}$ is a $\Sigma^{\text{DOD-alt}}$-theory. The *DOD alteration space* $\text{IS}(\Sigma^{\text{DOD-alt}})$ is the set of all DOD alteration states over $\Sigma^{\text{DOD-alt}}$.

An *RQS alteration state* is a design process state at the second meta-level, linked through a reflection relation to a requirement qualification set state at the first meta-level. It not only captures information as defined by the alteration vocabulary, but also the following types of information that apply only to the current RQS alteration state:

- epistemic information about the design requirement information included in the current requirement qualification set,
- goals to deduce design requirement information from the current requirement qualification set,
- information about which alterations are proposed, selected or rejected for application to the current requirement qualification set,
- assertions and retractions of assumptions about design requirement information for the current requirement qualification set,
- information about which requirement qualification set (to be retrieved from the history of the design process) is to replace the current requirement qualification set,
- queries for requirement qualification sets.

In addition, an RQS alteration state captures the following types of information:

- information about the composition of requirement qualification set alterations,
- assessments of requirement qualification sets.

As for design object description alterations, it is assumed for the sake of simplicity that each alteration is a set of modifications.

Definition 4.4.6. (RQS Alteration Vocabulary) Let $\Sigma^{alt} = (\mathbf{S}^{alt}, \mathbf{C}^{alt}, \mathbf{F}^{alt}, \mathbf{P}^{alt})$ be an alteration vocabulary. An *RQS alteration vocabulary* based on Σ^{alt} is an order-sorted signature $\Sigma^{RQS\text{-}alt} = (\mathbf{S}^{RQS\text{-}alt}, \mathbf{C}^{RQS\text{-}alt}, \mathbf{F}^{RQS\text{-}alt}, \mathbf{P}^{RQS\text{-}alt})$, which is is an extension of Σ^{alt}, and where the sorts $\mathbf{S}^{RQS\text{-}alt}$, functions $\mathbf{F}^{RQS\text{-}alt}$, and predicates $\mathbf{P}^{RQS\text{-}alt}$ are defined as follows (see Tables 4.12 to 4.14):

Table 4.12. Standard sorts within $\mathbf{S}^{RQS\text{-}alt}$.

Sort	*Explanation*
RQS-MODIFICATION	Elementary modifications of requirement qualification sets.
RQS-ALTERATION	Alterations (i.e., sets of modifications) of requirement qualification sets. RQS-MODIFICATION ≤ RQS-ALTERATION holds.

Table 4.13. Standard functions within $\mathbf{F}^{RQS\text{-}alt}$.

Function	*Explanation*
ADDITION-OF: DESIGN-REQUIREMENT-INFO-ELEMENT → RQS-MODIFICATION	Modification to the current requirement qualification set, involving the addition of a given design requirement information element.
DELETION-OF: DESIGN-REQUIREMENT-INFO-ELEMENT → RQS-MODIFICATION	Modification to the current requirement qualification set, involving the deletion of a given design requirement information element.

Table 4.14. Standard predicates within $\mathbf{P}^{RQS\text{-}alt}$.

Predicate	*Explanation*
IS-CURRENTLY-INCLUDED: DESIGN-REQUIREMENT-INFO-ELEMENT × SIGN	Design requirement information included in the current requirement qualification set.
IS-PART-OF-DEDUCTIVE-RQS-REFINEMENT-FOCUS: DESIGN-REQUIREMENT-INFO-ELEMENT × SIGN	Design requirement information to be deduced from the current requirement qualification set.
INCLUDES-RQS-MODIFICATION: RQS-ALTERATION × RQS-MODIFICATION	Statement that the given requirement qualification set alteration includes the given modification.
IS-PROPOSED-ALTERATION: RQS-ALTERATION	A proposed alteration to the current set.
IS-REJECTED-ALTERATION: RQS-ALTERATION	A rejected alteration to the current set.
IS-SELECTED-RQS-ALTERATION: RQS-ALTERATION	A selected alteration to the current set.
IS-TO-BE-ASSERTED: DESIGN-REQUIREMENT-INFO-ELEMENT × SIGN	Design requirement information to be added to the current requirement qualification set.

Predicate	*Explanation*
IS-TO-BE-RETRACTED: DESIGN-REQUIREMENT-INFO-ELEMENT × SIGN	Design requirement information to be deleted from the current requirement qualification set.
IS-REPLACEMENT-FOR-CURRENT-RQS: RQS-NAME	Statement that the given requirement qualification set is to become the current set.
IS-INCLUDED-IN-WHICH-RQSS: DESIGN-REQUIREMENT-INFO-ELEMENT × SIGN	Query for requirement qualification sets that include the given design requirement information.
INCLUDES-WHICH-DESIGN-REQUIREMENT-INFORMATION: RQS-NAME	Query for the design requirement information included in the given requirement qualification set.
CAN-BE-FULFILLED: RQS-NAME	Statement that the given requirement qualification set can be fulfilled (cf. Definition 3.6.5).
IS-INCONSISTENT: RQS-NAME	Statement that the given requirement qualification set is inconsistent (cf. Definition 3.6.5).
IS-AMBIGUOUS: RQS-NAME	Statement that the given requirement qualification set is ambiguous (cf. Definition 3.6.5).
IS-IMPRECISE: RQS-NAME	Statement that the given requirement qualification set is imprecise (cf. Definition 3.6.5).
IS-INCOMPLETE: RQS-NAME	Statement that the given requirement qualification set is incomplete (cf. Definition 3.6.5).

Definition 4.4.7. (RQS Alteration State, Theory, and Space) Let $\Sigma^{\text{RQS-alt}}$ be an RQS alteration vocabulary. An *RQS alteration state* over $\Sigma^{\text{RQS-alt}}$ is a partial $\Sigma^{\text{RQS-alt}}$-model. An *RQS alteration theory* over $\Sigma^{\text{RQS-alt}}$ is a $\Sigma^{\text{RQS-alt}}$-theory. The *RQS alteration space* $\text{IS}(\Sigma^{\text{RQS-alt}})$ is the set of all RQS alteration states over $\Sigma^{\text{RQS-alt}}$.

4.4.3 Overall Control States

Overall DOD control states, overall RQS control states, and overall co-ordination states have a number of types of information in common. In this section, a generic control vocabulary is defined, which is first extended to an overall DOD control vocabulary and then to an overall RQS control vocabulary and an overall co-ordination vocabulary.

The designer has to pave the road to success in a vast design space. For this purpose, the designer deliberately or unconsciously makes use of a design strategy, such as hierarchical decomposition, propose-and-revise, or trial-and-error. A *design strategy* is a plan (often of a heuristic nature) for exploring the design space, which is intended to improve the chance that the design process comes to a satisfactory solution, given the available design resources.

There are several (AI in design) researchers who acknowledge the usefulness of design strategies. Lawson, to name but one example, claims that design strategies are necessary, as in design, “the solution is not just the logical outcome of the problem, and there is therefore no sequence of operations which will guarantee a result” [Lawson, 1997]. He observes that

"most design strategies seem to begin with a brief scanning of the problem as it appears initially," while also "elements of solutions rather than problems begin to emerge very early on in the process". In terms of our logical theory of design, this means that making alterations to requirement qualification sets and making alterations to design object descriptions are intertwined (rather than cascading) processes, which make up for complex design strategies.

A *control vocabulary* is a vocabulary for expressions used in the preparation of a design strategy, the instruction of alteration processes to follow a specific design strategy, and the evaluation of alteration processes against a specific design strategy. A control vocabulary defines the following types of information that apply only to the current overall control state:

- information about the identity of the current overall design process state (within a design process as a whole),
- information about the identity of the current control process state (within an alteration process),
- control process evaluation of an alteration process against a specific design strategy,
- information about the current control decision,
- queries for design process states.

In addition, a control vocabulary defines the following types of information:

- information about the control process plans underlying specific design strategies (that, for many practical cases, can be expressed using constructs such as '*if …then …*" and "*while … do …*".),
- information about the design strategy, design process results, control decision, or control process evaluation included in a specific design process state,
- information about the successors and predecessors of specific design process states.

Definition 4.4.8. (Control Vocabulary) Let $\Sigma^{alt} = (\mathbf{S}^{alt}, \mathbf{C}^{alt}, \mathbf{F}^{alt}, \mathbf{P}^{alt})$ be an alteration vocabulary. A *control vocabulary* based on Σ^{alt} is an order-sorted signature $\Sigma^{control} = (\mathbf{S}^{control}, \mathbf{C}^{control}, \mathbf{F}^{control}, \mathbf{P}^{control})$, where the sorts $\mathbf{S}^{control}$, constants $\mathbf{C}^{control}$, functions $\mathbf{F}^{control}$, and predicates $\mathbf{P}^{control}$ are defined as follows (see Tables 4.15 to 4.18):

Table 4.15. Standard sorts within $\mathbf{S}^{control}$.

Sort	*Explanation*
DESIGN-PROCESS-RESULTS-INFO-ELEMENT	Well-formed atoms over the alteration vocabulary Σ^{alt}.
DESIGN-PROCESS-RESULTS-FORMULA	The set of well-formed formulas over the domain object vocabulary Σ^{alt}. DESIGN-PROCESS-RESULTS-INFO-ELEMENT ≤ DESIGN-PROCESS-RESULTS-INFO-FORMULA holds.
SIGN	Truth values in a three-valued logic.
CONTROL-PROCESS-PLAN	Possible control process plans for design strategies.

Sort	*Explanation*
DESIGN-STRATEGY-NAME	Possible names of design strategies (for ease of reference).
DESIGN-STRATEGY	Possible design strategies. DESIGN-STRATEGY-NAME ≤ DESIGN-STRATEGY and CONTROL-PROCESS-PLAN ≤ DESIGN-STRATEGY both hold.
SELECTION-CRITERION	Possible selection criteria to be used in control process plans.
SELECTION-CRITERION-LIST	Possible lists of selection criteria to be used in the formulation of design strategies. SELECTION-CRITERION ≤ SELECTION-CRITERION-LIST holds.
CONTROL-APPROACH	Possible control approaches to be used in control process plans.
DESIGN-PROCESS-STATE	Possible states of a design process.
CONTROL-PROCESS-EVALUATION	Possible (results of) evaluations of control processes against design strategies.
CONTROL-DECISION	Possible decisions about whether or not to continue the control process of concern, and if so, doing what.

The following control decisions can be expressed, which are all related to making alterations to requirement qualification sets or design object descriptions:

- *deductive refinement* of the current requirement qualification set or design object description;
- *modification* of the current requirement qualification set or design object description;
- *replacement* of the current requirement qualification set or design object description;
- *query and retrieval* of requirement qualification sets or design object descriptions from the history of the design process;
- *termination* of the RQS alteration process or DOD alteration process.

The following control approaches are based on Treur's (non-exhaustive) list of specific uses of reasoning about design requirements [Treur, 1991] and can be applied to generate modifications to both requirement qualification sets and design object descriptions:

- a *transformation* of the available information into different but equivalent information (e.g., transformation of the equation $x^2 + 4x - 5 = 0$ into $(x - 1)(x + 5) = 0$);
- a *translation* of the available information into another language (e.g., translation of the equation $(x - 1)(x + 5) = 0$ into $(y - 3)(y + 3) = 0$, plus the extra equation $y = x + 2$);
- a *decomposition* of the available information into new information (e.g., decomposition of global requirements of a software system into more detailed requirements);
- a *composition* of new information from the available information (e.g., composition of a software system from a set of sub-systems and a set of sub-system interfaces);
- a *reduction* of the available information (e.g., removing a design requirement);
- an *extension* of the available information (e.g., adding a design requirement).

Table 4.16. Standard constants within $\mathbf{C}^{\text{control}}$.

Constant	*Explanation*
'IsTrue': SELECTION-CRITERION	A tautology (i.e., something which is always true).
'IsFalse': SELECTION-CRITERION	A contradiction (i.e., something which is always false).
'Nil': SELECTION-CRITERION-LIST	The empty list of selection criteria.
'Pos', 'Neg', 'Unk': SIGN	The truth values *true*, *false*, and *unknown/undefined*.
'Transformation', 'Translation', 'Decomposition', 'Composition', 'Reduction', 'Extension': CONTROL-APPROACH	Possible control approaches: a transformation, a translation, a decomposition, a composition, a reduction, and an extension.
'InitialState': DESIGN-PROCESS-STATE	The initial state of an alteration process.
'Incomplete', 'Succeeded', 'Failed': CONTROL-PROCESS-EVALUATION	Possible evaluations of the termination of a control process against a design strategy: inconclusive due to incomplete information, success, and failure.
'DeductiveRefinement', 'Modification', 'Replacement', 'QueryAndRetrieval', 'Termination': CONTROL-DECISION	Possible control decisions: deductive refinement, modification, replacement, query-and-retrieval, and termination.

Table 4.17. Standard functions within $\mathbf{F}^{\text{control}}$.

Function	*Explanation*
'Not': DESIGN-PROCESS-RESULTS-INFO-FORMULA → DESIGN-PROCESS-RESULTS-INFO-FORMULA	Negation of the given well-formed formula.
'And': DESIGN-PROCESS-RESULTS-INFO-FORMULA × DESIGN-PROCESS-RESULTS-INFO-FORMULA → DESIGN-PROCESS-RESULTS-INFO-FORMULA	Conjunction of the two given well-formed formulas.
'Or': DESIGN-PROCESS-RESULTS-INFO-FORMULA × DESIGN-PROCESS-RESULTS-INFO-FORMULA → DESIGN-PROCESS-RESULTS-INFO-FORMULA	Disjunction of the two given well-formed formulas.
'Implies': DESIGN-PROCESS-RESULTS-INFO-FORMULA × DESIGN-PROCESS-RESULTS-INFO-FORMULA → DESIGN-PROCESS-RESULTS-INFO-FORMULA	Implication of the two given well-formed formulas.
'Dot', '.': SELECTION-CRITERION × SELECTION-CRITERION-LIST → SELECTION-CRITERION-LIST	A selection criterion list that has the given selection criterion as its head and the given selection criterion list as its tail.
'ContinueWith': RQS-NAME × DOD-NAME → CONTROL-PROCESS-PLAN	The control process plan to continue with the given requirement qualification set and design object description as the current set and current description, respectively.

Function	*Explanation*
'IfThen': DESIGN-PROCESS-RESULTS-INFO-FORMULA × CONTROL-PROCESS-PLAN → CONTROL-PROCESS-PLAN	The control process plan involving that, if the given condition on design process results information holds, then the given control process plan is executed.
'IfThenElse': DESIGN-PROCESS-RESULTS-INFO-FORMULA × CONTROL-PROCESS-PLAN × CONTROL-PROCESS-PLAN → CONTROL-PROCESS-PLAN	The control process plan involving that, if the given condition on design process results information holds, the first given control process plan is executed, otherwise the second given control process plan is executed.
'WhileDo': DESIGN-PROCESS-RESULTS-INFO-FORMULA × CONTROL-PROCESS-PLAN → CONTROL-PROCESS-PLAN	The control process plan involving that, as long as the given condition on design process results information holds, the given control process plan is executed.
'RepeatUntil': CONTROL-PROCESS-PLAN × DESIGN-PROCESS-RESULTS-INFO-FORMULA → CONTROL-PROCESS-PLAN	The control process plan involving that the given control process plan is executed until the given condition on design process results information holds.
'DoInSequence': CONTROL-PROCESS-PLAN × CONTROL-PROCESS-PLAN → CONTROL-PROCESS-PLAN	The control process plan involving that the first given control process plan is executed followed by the second given control process plan.
'DoInParallel': CONTROL-PROCESS-PLAN × CONTROL-PROCESS-PLAN → CONTROL-PROCESS-PLAN	The control process plan involving that the two given control process plans are executed simultaneously.
'ApplyCriteriaForRetrievalApproach': SELECTION-CRITERION-LIST × CONTROL-APPROACH → CONTROL-PROCESS-PLAN	The control process plan involving that the given control approach is applied by retrieving those requirement qualification sets (or design object descriptions) that satisfy the given selection criteria.
'ApplyCriteriaForModificationApproach': SELECTION-CRITERION-LIST × CONTROL-APPROACH → CONTROL-PROCESS-PLAN	The control process plan involving that the given control approach is applied by modifying those elements that satisfy the given selection criteria.
'Succ': DESIGN-PROCESS-STATE → DESIGN-PROCESS-STATE	The design process state succeeding a given design process state.

Table 4.18. Standard predicates within $\mathbf{P}^{\text{control}}$.

Predicate	*Explanation*
IS-DEFINED-AS: DESIGN-STRATEGY-NAME × CONTROL-PROCESS-PLAN	Statement about the unique control process plan corresponding to the given design strategy name.
IS-CURRENT-OVERALL-DESIGN-PROCESS-STATE: DESIGN-PROCESS-STATE	Statement about the identity of the current overall design process state.
INCLUDES-DESIGN-STRATEGY: DESIGN-PROCESS-STATE × DESIGN-STRATEGY	Statement about the design strategy included in a given design process state.
IS-CURRENT-CONTROL-PROCESS-STATE: DESIGN-PROCESS-STATE	Statement about the identity of the current control process state.

Predicate	*Explanation*
IS-CURRENT-CONTROL-PROCESS-PLAN: CONTROL-PROCESS-PLAN	Statement about the contents of the current control process plan.
HAS-DOD-CONTROL-PROCESS-EVALUATION: DESIGN-STRATEGY × CONTROL-PROCESS-EVALUATION	The control process evaluation of the overall DOD control process against the given design strategy.
HAS-RQS-CONTROL-PROCESS-EVALUATION: DESIGN-STRATEGY × CONTROL-PROCESS-EVALUATION	The control process evaluation of the overall RQS control process against the given design strategy.
INCLUDES-DESIGN-PROCESS-RESULTS-INFORMATION: DESIGN-PROCESS-STATE × DESIGN-PROCESS-RESULTS-INFO-ELEMENT × SIGN	Statement about the design process results information included in the given design process state.
HAS-SUCCEEDING-DESIGN-PROCESS-STATE: DESIGN-PROCESS-STATE × DESIGN-PROCESS-STATE	Statement about the design process state succeeding a given design process state.
INCLUDES-CONTROL-DECISION: DESIGN-PROCESS-STATE × CONTROL-DECISION	Statement about the control decision in a given design process state.
INCLUDES-CONTROL-PROCESS-EVALUATION: DESIGN-PROCESS-STATE × CONTROL-PROCESS-EVALUATION	Statement about the control process evaluation in a given design process state.
IS-CURRENT-CONTROL-DECISION: CONTROL-DECISION	Statement that the given control decision has been taken in the current overall control process state.
INCLUDES-WHICH-DESIGN-STRATEGY: DESIGN-PROCESS-STATE	Query for the design strategy in the given design process state.
IS-INCLUDED-IN-WHICH-DESIGN-PROCESS-STATES: DESIGN-STRATEGY	Query for the design process states that include the given design strategy.
INCLUDES-WHICH-DESIGN-PROCESS-RESULTS-INFORMATION: DESIGN-PROCESS-STATE	Query for the design process results information in the given design process state.
IS-INCLUDED-IN-WHICH-DESIGN-PROCESS-STATES: DESIGN-PROCESS-RESULTS-INFO-ELEMENT × SIGN	Query for the design process states that include the given design process results information.
IS-PRECEDED-BY-WHICH-DESIGN-PROCESS-STATE: DESIGN-PROCESS-STATE	Query for the design process state preceding the given design process state.
IS-SUCCEEDED-BY-WHICH-DESIGN-PROCESS-STATE: DESIGN-PROCESS-STATE	Query for the design process state succeeding the given design process state.
INCLUDES-WHICH-CONTROL-DECISION DESIGN-PROCESS-STATE	Query for the control decision in the given design process state.
IS-DECISION-IN-WHICH-DESIGN-PROCESS-STATES: CONTROL-DECISION	Query for the design process states that include the given control decision.
INCLUDES-WHICH-CONTROL-PROCESS-EVALUATION: DESIGN-PROCESS-STATE	Query for the control process evaluation in the given design process state.
IS-EVALUATION-OF-WHICH-DESIGN-PROCESS-STATES: CONTROL-DECISION	Query for the design process states that include the given control process evaluation.

An *overall DOD control state* is a design process state at the second meta-level, which is linked through a reflection relation to a DOD alteration state at the second meta-level. It not only captures information as defined by the control vocabulary, but also the following types of information:

- queries for overall DOD control states,
- epistemic information about the DOD modification process information included in specific design process states.

Definition 4.4.9. (Overall DOD Control Vocabulary) Let $\Sigma^{\text{DOD-alt}} = (\mathbf{S}^{\text{DOD-alt}}, \mathbf{C}^{\text{DOD-alt}}, \mathbf{F}^{\text{DOD-alt}}, \mathbf{P}^{\text{DOD-alt}})$ be a DOD alteration vocabulary, and $\Sigma^{\text{control}} = (\mathbf{S}^{\text{control}}, \mathbf{C}^{\text{control}}, \mathbf{F}^{\text{control}}, \mathbf{P}^{\text{control}})$ a control vocabulary. An *overall DOD control vocabulary* based on $\langle \Sigma^{\text{DOD-alt}}, \Sigma^{\text{control}} \rangle$ is an order-sorted signature $\Sigma^{\text{DOD-control}} = (\mathbf{S}^{\text{DOD-control}}, \mathbf{C}^{\text{DOD-control}}, \mathbf{F}^{\text{DOD-control}}, \mathbf{P}^{\text{DOD-control}})$, which extends Σ^{control}, and where the sorts $\mathbf{S}^{\text{DOD-control}}$ and predicates $\mathbf{P}^{\text{DOD-control}}$ are defined as follows (see Tables 4.19 and 4.20):

Table 4.19. Standard sorts within $\mathbf{S}^{\text{DOD-control}}$.

Sort	*Explanation*
DOD-MODIFICATION-PROCESS-INFO-ELEMENT	Well-formed atoms over the DOD alteration vocabulary $\Sigma^{\text{DOD-alt}}$.

Table 4.20. Standard predicates within $\mathbf{P}^{\text{DOD-control}}$.

Predicate	*Explanation*
INCLUDES-DOD-MODIFICATION-PROCESS-INFORMATION: DESIGN-PROCESS-STATE × DOD-MODIFICATION-PROCESS-INFO-ELEMENT × SIGN	Statement about DOD modification process information included in the given design process state.
INCLUDES-WHICH-DOD-MODIFICATION-PROCESS-INFORMATION: DESIGN-PROCESS-STATE	Query for the DOD modification process information included in the given design process state.
IS-INCLUDED-IN-WHICH-DESIGN-PROCESS-STATES: DOD-MODIFICATION-PROCESS-INFO-ELEMENT × SIGN	Query for the design process states that include the given DOD modification process information.

Definition 4.4.10. (Overall DOD Control State and Space) Let $\Sigma^{\text{DOD-control}}$ be an overall DOD control vocabulary. An *overall DOD control state* over $\Sigma^{\text{DOD-control}}$ is a partial $\Sigma^{\text{DOD-control}}$-model. The *overall DOD control space* $\text{IS}(\Sigma^{\text{DOD-control}})$ is the set of all overall DOD control states over $\Sigma^{\text{DOD-control}}$.

An *overall RQS control state* is a design process state at the third meta-level, linked through a reflection relation to a RQS alteration state at the second meta-level. It not only

captures information as defined by the control vocabulary, but also the following types of information:

- queries for overall RQS control states,
- epistemic information about the RQS modification process information included in specific design process states.

Definition 4.4.11. (Overall RQS Control Vocabulary) Let $\Sigma^{\text{RQS-alt}} = (\mathbf{S}^{\text{RQS-alt}}, \mathbf{C}^{\text{RQS-alt}}, \mathbf{F}^{\text{RQS-alt}}, \mathbf{P}^{\text{RQS-alt}})$ be an RQS alteration vocabulary, and $\Sigma^{\text{control}} = (\mathbf{S}^{\text{control}}, \mathbf{C}^{\text{control}}, \mathbf{F}^{\text{control}}, \mathbf{P}^{\text{control}})$ a control vocabulary. An *overall RQS control vocabulary* based on $\langle \Sigma^{\text{RQS-alt}}, \Sigma^{\text{control}} \rangle$ is an order-sorted signature $\Sigma^{\text{RQS-control}} = (\mathbf{S}^{\text{RQS-control}}, \mathbf{C}^{\text{RQS-control}}, \mathbf{F}^{\text{RQS-control}}, \mathbf{P}^{\text{RQ-control}})$, which is an extension of Σ^{control}, and where the sorts $\mathbf{S}^{\text{RQS-control}}$ and predicates $\mathbf{P}^{\text{RQS-control}}$ are defined as follows (see Tables 4.21 and 4.22):

Table 4.21. Standard sorts within $\mathbf{S}^{\text{RQS-control}}$.

Sort	*Explanation*
RQS-MODIFICATION-PROCESS-INFO-ELEMENT	Well-formed atoms over the RQS alteration vocabulary $\Sigma^{\text{RQS-alt}}$.

Table 4.22. Standard predicates within $\mathbf{P}^{\text{RQS-control}}$.

Predicate	*Explanation*
INCLUDES-RQS-MODIFICATION-PROCESS-INFORMATION: DESIGN-PROCESS-STATE × RQS-MODIFICATION-PROCESS-INFO-ELEMENT × SIGN	Statement about the RQS modification process information included in the given design process state.
INCLUDES-WHICH-RQS-MODIFICATION-PROCESS-INFORMATION: DESIGN-PROCESS-STATE	Query for the RQS modification process information included in the given design process state.
IS-INCLUDED-IN-WHICH-DESIGN-PROCESS-STATES: RQS-MODIFICATION-PROCESS-INFO-ELEMENT × SIGN	Query for the design process states that include the given RQS modification process information.

Definition 4.4.12. (Overall RQS Control State and Space) Let $\Sigma^{\text{RQS-control}}$ be an overall RQS control vocabulary. An *overall RQS control state* over $\Sigma^{\text{RQS-control}}$ is a partial $\Sigma^{\text{RQS-control}}$-model. The *overall RQS control space* $\text{IS}(\Sigma^{\text{RQS-control}})$ is the set of all overall RQS control states over $\Sigma^{\text{RQS-control}}$.

An *overall co-ordination state* is a design process state at the third meta-level. It not only captures information as defined by the control vocabulary, but also information about the definitions of design process objectives, information about which of the given design object objectives need to be satisfied, and information about which design process objectives are satisfied or violated.

Definition 4.4.13. (Overall Co-ordination Vocabulary) Let $\Sigma^{control} = (\mathbf{S}^{control}, \mathbf{C}^{control}, \mathbf{F}^{control}, \mathbf{P}^{control})$ be a control vocabulary. Then an *overall co-ordination vocabulary* based on $\Sigma^{control}$ is an order-sorted signature $\Sigma^{coord} = (\mathbf{S}^{coord}, \mathbf{C}^{coord}, \mathbf{F}^{coord}, \mathbf{P}^{coord})$, which is an extension of $\Sigma^{control}$, and where the sorts $\mathbf{S}^{coord}$, constants $\mathbf{C}^{coord}$, functions $\mathbf{F}^{coord}$, and predicates $\mathbf{P}^{coord}$ are defined as follows (see Tables 4.23 to 4.26):

Table 4.23. Standard sorts within $\mathbf{S}^{coord}$.

Sort	*Explanation*
PROCESS-OBJECTIVE-NAME	Possible names of process objectives (for ease of reference).
PROCESS-OBJECTIVE	Possible process objectives. PROCESS-OBJECTIVE-NAME ≤ PROCESS-OBJECTIVE and DESIGN-PROCESS-RESULTS-INFO-FORMULA ≤ PROCESS-OBJECTIVE both hold.
QUALIFIED-PROCESS-OBJECTIVE-NAME	Possible names of qualified process objectives (for ease of reference).
INTEGER	Integers (to be used in the formulation of numeric constraints).
NUMERIC-CONSTRAINT	Possible numeric constraints (to be used in the formulation of qualifications of process objectives).
QUALIFICATION	Possible qualifications of process objectives.
PROCESS-OBJECTIVE-LIST	Possible lists of process objectives, which are used in the formulation of qualified process objectives. PROCESS-OBJECTIVE ≤ PROCESS-OBJECTIVE-LIST holds.
QUALIFIED-PROCESS-OBJECTIVE-EXPRESSION	Possible expressions of qualified process objectives.
QUALIFIED-PROCESS-OBJECTIVE	Possible qualified process objectives. QUALIFIED-PROCESS-OBJECTIVE-NAME ≤ QUALIFIED-PROCESS-OBJECTIVE and QUALIFIED-PROCESS-OBJECTIVE-EXPRESSION ≤ QUALIFIED-PROCESS-OBJECTIVE both hold.
DESIGN-PROCESS-OBJECTIVE	Possible design process objectives. PROCESS-OBJECTIVE ≤ DESIGN-PROCESS-OBJECTIVE and QUALIFIED-PROCESS-OBJECTIVE ≤ DESIGN-PROCESS-OBJECTIVE both hold.

Table 4.24. Standard constants within $\mathbf{C}^{coord}$.

Constant	*Explanation*
‘Zero’, ‘0’: INTEGER	The integer zero.
‘Any’: NUMERIC-CONSTRAINT	Constraint denoting “one or more.”
‘AllPossible’: NUMERIC-CONSTRAINT	Constraint denoting “zero or more.”
‘Every’: QUALIFICATION	Qualification denoting that every item (from a set) counts.
‘Nil’, ‘[]’: PROCESS-OBJECTIVE-LIST	The empty list of process objectives.

Table 4.25. Standard functions within $\mathbf{F}^{coord}$.

Function	*Explanation*
'Succ', '+1': INTEGER → INTEGER	The successor of an integer.
'AtLeast': INTEGER → NUMERIC-CONSTRAINT	Constraint denoting "at least *the given number*."
'Exactly': INTEGER → NUMERIC-CONSTRAINT	Constraint denoting "exactly *the given number*."
'AtMost': INTEGER → NUMERIC-CONSTRAINT	Constraint denoting "at most *the given number*."
'AtRandom': NUMERIC-CONSTRAINT → QUALIFICATION	Qualification denoting that the order of items in a list that is subject to the given constraint is insignificant.
'InPreferredOrder': NUMERIC-CONSTRAINT → QUALIFICATION	Qualification denoting that the order of items in a list that is subject to the given constraint is significant.
'Dot': PROCESS-OBJECTIVE × PROCESS-OBJECTIVE-LIST → PROCESS-OBJECTIVE-LIST	The process objective list that has the given process objective as its head, and the given process objective list as its tail.
'QualifiedProcessObjectiveExpr': QUALIFICATION × PROCESS-OBJECTIVE-LIST → QUALIFIED-PROCESS-OBJECTIVE-EXPRESSION	The qualified process objective expression built from the given qualification and the given process objective list.

The predicate IS-TO-BE-SATISFIED can be used to point out the individual design process objectives that must be satisfied by the design process. For the sake of clarity, it is assumed that at least all qualified process objectives presented to the design process must be satisfied.

Table 4.26. Standard predicates within $\mathbf{P}^{coord}$.

Predicate	*Explanation*
IS-DEFINED-AS: PROCESS-OBJECTIVE-NAME × DESIGN-PROCESS-RESULTS-INFO-FORMULA	Statement about the unique well-formed formula over the alteration vocabulary that corresponds to the given process objective name.
IS-DEFINED-AS: QUALIFIED-PROCESS-OBJECTIVE-NAME × QUALIFIED-PROCESS-OBJECTIVE-EXPRESSION	Statement about the unique qualified process objective expression that corresponds to the given qualified process objective name.
IS-TO-BE-SATISFIED: DESIGN-PROCESS-OBJECTIVE	Statement that the given design process objective must be satisfied.
IS-PART-OF-DESIGN-PROCESS-RESULTS: DESIGN-PROCESS-RESULTS-INFO-ELEMENT × SIGN	Statement that the given result has been produced by the design process.
IS-SATISFIED: DESIGN-PROCESS-OBJECTIVE	Statement that the given design process objective is satisfied (by the design process).
IS-VIOLATED: DESIGN-PROCESS-OBJECTIVE	Statement that the given design process objective is violated.
IS-DECIDED: DESIGN-PROCESS-OBJECTIVE	Statement that the given design process objective is known to be either satisfied or violated.

Definition 4.4.14. (Overall Co-ordination State and Space) Let Σ^{coord} be an overall co-ordination vocabulary. An *overall co-ordination state* over Σ^{coord} is a partial Σ^{coord}-model. The *overall co-ordination space* $IS(\Sigma^{coord})$ is the set of all overall co-ordination states over Σ^{coord}.

An *overall design control state* is a design process state that captures the strategic control information of a design process. It combines an overall design object description control state, an overall requirement qualification set control state, and an overall co-ordination state.

Definition 4.4.15. (Overall Design Control Vocabulary) Let $\Sigma^{DOD\text{-}control}$ be an overall DOD control vocabulary, let $\Sigma^{RQS\text{-}control}$ be an overall RQS control vocabulary, and let Σ^{coord} be an overall co-ordination vocabulary. An *overall design control vocabulary* based on $\langle\Sigma^{DOD\text{-}control}, \Sigma^{RQS\text{-}control}, \Sigma^{coord}\rangle$ is an order-sorted signature $\Sigma^{design\text{-}control} = \Sigma^{DOD\text{-}control} \cup \Sigma^{RQS\text{-}control} \cup \Sigma^{coord}$. That is, the four components of $\Sigma^{design\text{-}control}$ are each the union of the corresponding components of $\Sigma^{DOD\text{-}control}$, $\Sigma^{RQS\text{-}control}$, and Σ^{coord}:

- $S(\Sigma^{design\text{-}control}) = S(\Sigma^{DOD\text{-}control}) \cup S(\Sigma^{RQS\text{-}control}) \cup S(\Sigma^{coord})$.
- $C(\Sigma^{design\text{-}control}) = C(\Sigma^{DOD\text{-}control}) \cup C(\Sigma^{RQS\text{-}control}) \cup C(\Sigma^{coord})$.
- $F(\Sigma^{design\text{-}control}) = F(\Sigma^{DOD\text{-}control}) \cup F(\Sigma^{RQS\text{-}control}) \cup F(\Sigma^{coord})$.
- $P(\Sigma^{design\text{-}control}) = P(\Sigma^{DOD\text{-}control}) \cup P(\Sigma^{RQS\text{-}control}) \cup P(\Sigma^{coord})$.

Definition 4.4.16. (Overall Design Control State and Space) Let $\Sigma^{design\text{-}control}$ be an overall design control vocabulary. Then an *overall design control state* over $\Sigma^{design\text{-}control}$ is a partial $\Sigma^{design\text{-}control}$-model. The *overall design control space*, $IS(\Sigma^{design\text{-}control})$, is the set of all overall design control states over $\Sigma^{design\text{-}control}$.

4.5 Design Process Steps and Overall Design Traces

In this thesis, a design process is regarded as a series of design process states, which non-monotonically proceeds from the initial design process state to the final design process state. It carries out design activities and makes decisions about whether to continue or not, and if so, what to do next. From a logical point of view, a design activity involves acquiring new facts (e.g., a given requirement of the volume of the design object), introducing new assumptions (e.g., about the length of the design object), retracting existing assumptions (e.g., a requirement about the height of the design object), establishing goals for the deduction of implicit facts (e.g., the area of the design object), making deductions, and so on.

A *design decision* is a decision made about the next design step to take, intended to add detail and/or remove a *design conflict*: anything that is in the way of reaching a design solution. Removing a design conflict often requires making new design decisions or undoing earlier decisions while hopefully avoiding the introduction of new conflicts. A design decision may be subject to strategic considerations, such as whether or not to design from scratch.

A *design step* (or design move, design change, or design revision) is a transition of one design state into another, as a result of performing a design activity. There are different types of transitions, corresponding to the different types of design states.

Figure 4.3 shows an example of a sequence of transitions of design object description states and requirement qualification set states. In total, the figure shows fourteen steps. Nine steps are in the design object description state space, including one backtracking step from the third design object description back to the original design object description. Five steps are in the requirement qualification set state space, including one backtracking step from the third requirement qualification set back to the first requirement qualification set generated.

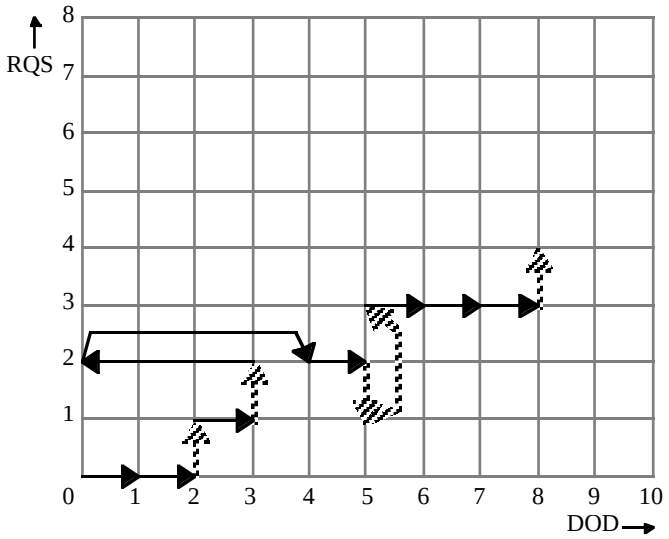


FIGURE 4.3. A fourteen-step sequence of DOD state transitions and RQS state transitions.

Using partial models to model design states, transitions of one design state into another are modelled by means of pairs of partial models. That is, each of the transitions transforms information states (i.e., possibly incomplete descriptions of different situations) into other information states. Figure 4.4 gives an overview of the types of transitions defined in this section. The transition types of which the name ends on *downward reflection* or *upward reflection* are more detailed interpretations of the five reflection relations introduced just before Figure 4.2 in Section 4.4.

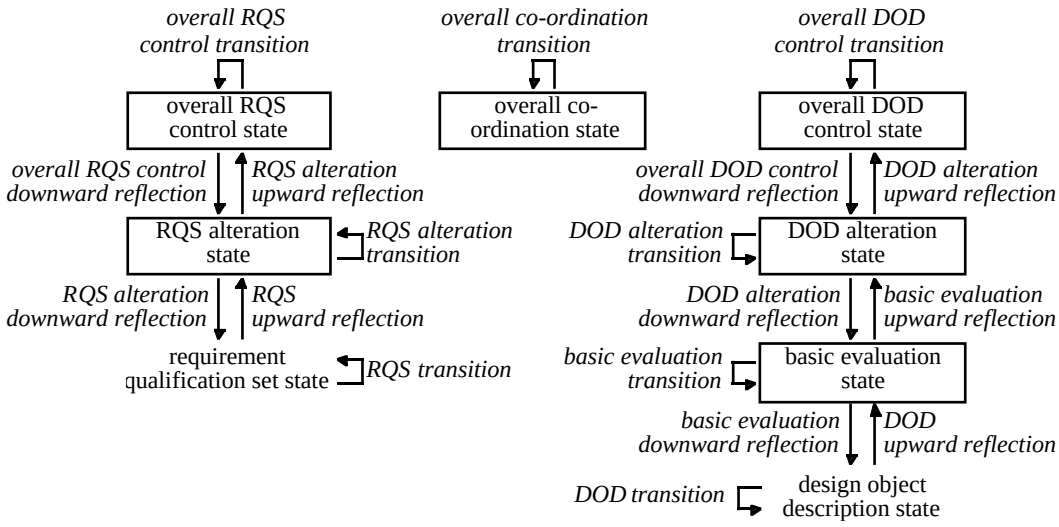


FIGURE 4.4. Types of transitions between design states.

Table 4.27 shows the vocabularies and theories that, in the remainder of this section, are assumed to be given and fixed.

Table 4.27. Vocabularies and theories used in the description of transitions.

Concept	*Meaning*
Σ^{DO}	Domain object vocabulary.
Φ^{DO}	Domain object theory over Σ^{DO} (i.e., a Σ^{DO}-theory).
$\Sigma^{basic\text{-}eval}$	Basic evaluation vocabulary.
Σ^{RQ}	Design requirement vocabulary.
Φ^{RQ}	Design requirement theory over Σ^{RQ} (i.e., a Σ^{RQ}-theory).
$\Sigma^{DOD\text{-}alt}$	DOD alteration vocabulary.
$\Sigma^{RQS\text{-}alt}$	RQS alteration vocabulary.
$\Sigma^{DOD\text{-}control}$	Overall DOD control vocabulary.
$\Sigma^{RQS\text{-}control}$	Overall RQS control vocabulary.
Σ^{coord}	Overall co-ordination vocabulary.

The transition type *DOD transition* models a design activity operating on the current design object description.

Definition 4.5.1. (DOD Transition) A *DOD transition* is a pair $\langle D, D' \rangle \in \text{IS}(\Sigma^{DO}) \times \text{IS}(\Sigma^{DO})$.

A DOD transition can be used to model the derivation of implicit domain object information from the explicit domain object information included in a design object description. (In such a case, for a DOD transition $\langle D, D' \rangle$, D' is a deductive refinement of D under Φ^{DO}.) For

example, suppose that a design object description D321 of a kitchen describes the length of the kitchen to be 6 metres, and the width of the kitchen to be 5 metres; it does not include any other information. The diagram of this design object description may read as follows:

{ HAS-VALUE(Kitchen1, Length(m), 6), HAS-VALUE(Kitchen1, Width(m), 5) }

Suppose further that there is knowledge available about the application domain, which relates the length, width and area of an object. A rule expressing this knowledge may read as follows (neglecting the sorts of variables, for the sake of readability):

$\forall o\ \forall u\ \forall l\ \forall w\ \forall a$ (
HAS-VALUE(o, Length(u), l) $\wedge$ HAS-VALUE(o, Width(u), w) $\wedge$ $a = l * w$) $\Rightarrow$
HAS-VALUE(o, Area(u^2), a))

Using this knowledge, a design object description D322 can be determined, of which the diagram reads as follows:

{ HAS-VALUE(Kitchen1, Length(m), 6), HAS-VALUE(Kitchen1, Width(m), 5),
HAS-VALUE(Kitchen1, Area(m^2), 30) }

Then ⟨D321, D322⟩ is a DOD transition where D322 is a deductive refinement of D321.

The transition type *DOD upward reflection* models the mapping of epistemic design object information included in the current design object description to basic evaluation information about the current design object description.

Definition 4.5.2. (DOD Upward Reflection) A *DOD upward reflection* is a pair $\langle D, E\rangle \in$ IS(Σ^{DO}) $\times$ IS($\Sigma^{basic\text{-}eval}$), such that the following conditions hold:

1. For all atoms $\varphi \in$ AT(Σ^{DO}), if $D \models \varphi$, then $E \models$ IS-CURRENTLY-INCLUDED('φ', 'Pos').
2. For all atoms $\varphi \in$ AT(Σ^{DO}), if $D \not\models \varphi$, then $E \models$ IS-CURRENTLY-INCLUDED('φ', 'Neg').
3. There is no other non-tautological sentence $\phi \in$ WFF($\Sigma^{basic\text{-}eval}$) such that $E \models \phi$.

For example, if AT(Σ^{DO}) = {p_1, p_2, q_1, q_2, q_3} and D is a model of p_1, ($\neg p_2$), and q_2, then E is a model of IS-CURRENTLY-INCLUDED('p_1', 'Pos'), IS-CURRENTLY-INCLUDED('p_2', 'Neg'), and IS-CURRENTLY-INCLUDED('q_2', 'Pos').

The transition type *basic evaluation downward reflection* models the mapping of basic evaluation information about the current design object description to assumptions on design object information included in the current design object description.

Definition 4.5.3. (Basic Evaluation Downward Reflection) A *basic evaluation downward reflection* is a pair $\langle E, D\rangle \in$ IS($\Sigma^{basic\text{-}eval}$) $\times$ IS(Σ^{DO}), such that the following conditions hold:

1. For all atoms $\varphi \in \text{AT}(\Sigma^{DO})$, if $E \models \text{IS-TO-BE-ASSERTED}(\text{`}\varphi\text{'}, \text{`Pos'})$, then $D \models \varphi$.
2. For all atoms $\varphi \in \text{AT}(\Sigma^{DO})$, if $E \models \text{IS-TO-BE-ASSERTED}(\text{`}\varphi\text{'}, \text{`Neg'})$, then $D \models \neg\varphi$.
3. There is no other non-tautological sentence $\phi \in \text{WFF}(\Sigma^{DO})$ such that $D \models \phi$.

Definition 4.5.3 can be extended to account for targets for the deductive refinement of the current design object description. A target is set on an atom $\varphi \in \text{AT}(\Sigma^{DO})$ if $E \models \text{IS-PART-OF-DEDUCTIVE-DOD-REFINEMENT-FOCUS}(\text{`}\varphi\text{'}, \text{`Pos'})$ or $E \models \text{IS-PART-OF-DEDUCTIVE-DOD-REFINEMENT-FOCUS}(\text{`}\varphi\text{'}, \text{`Neg'})$ holds. The interested reader is referred to Treur's work on meta-level architectures for dynamic control of reasoning, which addresses this type of reasoning [Treur, 1994].

Definition 4.5.3 can be further extended such that the current design object description is only updated, which is a more intuitive approach. This requires the transition to be reformulated as a triplet $\langle E, D, D' \rangle \in \text{IS}(\Sigma^{\text{basic-eval}}) \times \text{IS}(\Sigma^{DO}) \times \text{IS}(\Sigma^{DO})$, where D is the design object description before processing the information from E, and D' is the design object description after processing the information from E. Also, the conditions posed by Definition 4.5.3 should be replaced by the following, which must hold for all atoms $\varphi \in \text{AT}(\Sigma^{DO})$:

1. If $E \models \text{IS-TO-BE-ASSERTED}(\text{`}\varphi\text{'}, \text{`Pos'})$, then $D'(\varphi) = 1$.
2. If $E \models \text{IS-TO-BE-ASSERTED}(\text{`}\varphi\text{'}, \text{`Neg'})$, then $D'(\varphi) = 0$.
3. If $E \models \text{IS-TO-BE-RETRACTED}(\text{`}\varphi\text{'}, \text{`Pos'})$ and $D(\varphi) = 1$, then $D'(\varphi) = u$.
4. If $E \models \text{IS-TO-BE-RETRACTED}(\text{`}\varphi\text{'}, \text{`Neg'})$ and $D(\varphi) = 0$, then $D'(\varphi) = u$.
5. Otherwise, $D'(\varphi) = D(\varphi)$.

The transition type *basic evaluation transition* models the results of design activities such as the assessment of design object descriptions against design requirements, the determination of a focus for the deductive refinement of the current design object description, the determination of default domain object information for specific design object descriptions, and the formulation of queries for design object descriptions.

Definition 4.5.4. (Basic Evaluation Transition) A *basic evaluation transition* over $\Sigma^{\text{basic-eval}}$ is a pair $\langle E, E' \rangle \in \text{IS}(\Sigma^{\text{basic-eval}}) \times \text{IS}(\Sigma^{\text{basic-eval}})$.

The transition type *basic evaluation upward reflection* models the mapping of epistemic basic evaluation information about the current design object description to design object description alteration information about the current design object description.

Definition 4.5.5. (Basic Evaluation Upward Reflection) A *basic evaluation upward reflection* is a pair $\langle E, A \rangle \in \text{IS}(\Sigma^{\text{basic-eval}}) \times \text{IS}(\Sigma^{\text{DOD-alt}})$, such that the following conditions hold:

1. For all atoms $\varphi \in \text{AT}(\Sigma^{\text{basic-eval}})$, if $E \models \varphi$, then $A \models \text{IS-PART-OF-DOD-MODIFICATION-RESULTS}(\text{`}\varphi\text{'}, \text{`Pos'})$.

2. For all atoms $\varphi \in \text{AT}(\Sigma^{\text{basic-eval}})$, if $E \nvDash \varphi$, then $A \vDash$ IS-PART-OF-DOD-MODIFICATION-RESULTS('φ', 'Neg').
3. There is no other non-tautological sentence $\phi \in \text{WFF}(\Sigma^{\text{DOD-alt}})$ such that $A \vDash \phi$.

The transition type *DOD alteration downward reflection* models the mapping of design object description alteration information about the current design object description to assumptions on basic evaluation information about the current design object description.

Definition 4.5.6. (DOD Alteration Downward Reflection) A *DOD alteration downward reflection* is a pair $\langle A, E\rangle \in \text{IS}(\Sigma^{\text{DOD-alt}}) \times \text{IS}(\Sigma^{\text{basic-eval}})$, such that the following conditions hold:

1. For all atoms $\varphi \in \text{AT}(\Sigma^{\text{basic-eval}})$, if $A \vDash$ IS-PART-OF-DOD-MODIFICATION-BASIS('φ', 'Pos'), then $E \vDash \varphi$.
2. For all atoms $\varphi \in \text{AT}(\Sigma^{\text{basic-eval}})$, if $A \vDash$ IS-PART-OF-DOD-MODIFICATION-BASIS('φ', 'Neg'), then $E \nvDash \varphi$.
3. There is no other non-tautological sentence $\phi \in \text{WFF}(\Sigma^{\text{basic-eval}})$ such that $E \vDash \phi$.

Analogous to the definition of a basic evaluation downward reflection (Definition 4.5.3), Definition 4.5.6 can be extended such that the current design object description modification basis is only updated, which is a more intuitive approach.

The transition type *RQS transition* models a design activity operating on the current requirement qualification set.

Definition 4.5.7. (RQS Transition) An *RQS transition* is a pair $\langle R, R'\rangle \in \text{IS}(\Sigma^{\text{RQ}}) \times \text{IS}(\Sigma^{\text{RQ}})$.

An RQS transition can be used to model the derivation of implicit design requirement information from the explicit design requirement information included in a requirement qualification set. (In such a case, for an RQS transition $\langle R, R'\rangle$, R' is a deductive refinement of R under Φ^{RQ}). For example, suppose that one of the customer's design requirements for the design of a house is that there must be a kitchen in the house, of which the area must be at most 36 m^2. The diagram of this requirement qualification set S123 may read as follows:

{ IS-TO-BE-SATISFIED(QualifiedRequirementExpr(Every,
  HAS-VALUE(Kitchen1, Type, Kitchen))),
 IS-TO-BE-SATISFIED(QualifiedRequirementExpr(Every,
  Exists(A, And(HAS-VALUE(Kitchen1, Area(m^2), A), A ≤ 36)))) }

Suppose further that there is knowledge available about the application domain, which states that if the house is to have a kitchen, then it should preferably be positioned on the north side of the house, as this will generally be the coolest part (on the northern hemisphere). A rule expressing this knowledge may read as follows (again neglecting the sorts of variables):

$\forall q\ \forall o$ (
IS-TO-BE-SATISFIED(
QualifiedRequirementExpr(q, HAS-VALUE(o, Type, Kitchen))) ⇒
IS-TO-BE-SATISFIED(
QualifiedRequirementExpr(AllPossible, HAS-VALUE(o, Orientation, North))))

Using this knowledge, a requirement qualification set S124 can be determined, of which the diagram reads as follows:

{ IS-TO-BE-SATISFIED(QualifiedRequirementExpr(Every,
HAS-VALUE(Kitchen1, Type, Kitchen))),
IS-TO-BE-SATISFIED(QualifiedRequirementExpr(AllPossible,
HAS-VALUE(Kitchen1, Orientation, North))),
IS-TO-BE-SATISFIED(QualifiedRequirementExpr(Every,
Exists(A, And(HAS-VALUE(Kitchen1, Area(m^2), A), A ≤ 36)))) }

Then ⟨S123, S124⟩ is an RQS transition where S124 is a deductive refinement of S123.

The transition type *RQS upward reflection* models the mapping of epistemic information about the current design object description to requirement qualification set alteration information about the current requirement qualification set.

Definition 4.5.8. (RQS Upward Reflection) An *RQS upward reflection* is a pair $\langle R, A\rangle \in \text{IS}(\Sigma^{\text{RQ}}) \times \text{IS}(\Sigma^{\text{RQS-alt}})$, such that the following conditions hold:

1. For all atoms $\varphi \in \text{AT}(\Sigma^{\text{RQ}})$, if $R \models \varphi$, then $A \models$ IS-CURRENTLY-INCLUDED(‘φ’, ‘Pos’).
2. For all atoms $\varphi \in \text{AT}(\Sigma^{\text{RQ}})$, if $R \not\models \varphi$, then $A \models$ IS-CURRENTLY-INCLUDED(‘φ’, ‘Neg’).
3. There is no other sentence $\phi \in \text{WFF}(\Sigma^{\text{RQS-alt}})$ such that $A \models \phi$.

The transition type *RQS alteration downward reflection* models the mapping of requirement qualification set alteration information about the current requirement qualification set to assumptions on design requirement information included in the current requirement qualification set.

Definition 4.5.9. (RQS Alteration Downward Reflection) An *RQS alteration downward reflection* is a pair $\langle A, R\rangle \in \text{IS}(\Sigma^{\text{RQS-alt}}) \times \text{IS}(\Sigma^{\text{RQ}})$, such that the following conditions hold:

1. For all atoms $\varphi \in \text{AT}(\Sigma^{\text{RQ}})$, if $A \models$ IS-TO-BE-ASSERTED(‘φ’, ‘Pos’), then $R \models \varphi$.
2. For all atoms $\varphi \in \text{AT}(\Sigma^{\text{RQ}})$, if $A \models$ IS-TO-BE-ASSERTED(‘φ’, ‘Neg’), then $R \models \neg\varphi$.
3. There is no other non-tautological sentence $\phi \in \text{WFF}(\Sigma^{\text{RQ}})$ such that $R \models \phi$.

Definition 4.5.9 can be extended to account for targets for the deductive refinement of the current requirement qualification set. A target is set on an atom $\varphi \in$ AT(Σ^{RQ}) if $A \models$ IS-PART-OF-DEDUCTIVE-RQS-REFINEMENT-FOCUS('φ', 'Pos') or $A \models$ IS-PART-OF-DEDUCTIVE-RQS-REFINEMENT-FOCUS('φ', 'Neg') holds. As before, the interested reader is referred to Treur's work on meta-level architectures for dynamic control of reasoning, which addresses this type of reasoning [Treur, 1994].

Definition 4.5.9 can be further extended such that the current requirement qualification set is only updated, which is a more intuitive approach. This requires the transition to be reformulated as a triplet $\langle A, R, R' \rangle \in$ IS($\Sigma^{RQS\text{-}alt}$) $\times$ IS(Σ^{RQ}) $\times$ IS(Σ^{RQ}), where R is the requirement qualification set before processing the information from A, and R' is the requirement qualification set after processing the information from A. Also, the conditions posed by Definition 4.5.9 should be replaced by the following, which must hold for all atoms $\varphi \in$ AT(Σ^{RQ}):

1. If $A \models$ IS-TO-BE-ASSERTED('φ', 'Pos'), then $R'(\varphi) = 1$.
2. If $A \models$ IS-TO-BE-ASSERTED('φ', 'Neg'), then $R'(\varphi) = 0$.
3. If $A \models$ IS-TO-BE-RETRACTED('φ', 'Pos') and $R(\varphi) = 1$, then $R'(\varphi) = u$.
4. If $A \models$ IS-TO-BE-RETRACTED('φ', 'Neg') and $R(\varphi) = 0$, then $R'(\varphi) = u$.
5. Otherwise, $R'(\varphi) = R(\varphi)$.

The transition type *DOD alteration transition* models the results of design activities such as the assessment of design object descriptions against requirement qualification sets, and the generation, rejection, and selection of proposals for making modifications to the current design object description.

Definition 4.5.10. (DOD Alteration Transition) A *DOD alteration transition* is a pair $\langle A, A' \rangle \in$ IS($\Sigma^{DOD\text{-}alt}$) $\times$ IS($\Sigma^{DOD\text{-}alt}$).

The transition type *DOD alteration upward reflection* models the mapping of epistemic information about the results of making alterations to design object descriptions to overall design object description control information.

Definition 4.5.11. (DOD Alteration Upward Reflection) A *DOD alteration upward reflection* is a pair $\langle A, C \rangle \in$ IS($\Sigma^{DOD\text{-}alt}$) $\times$ IS($\Sigma^{DOD\text{-}control}$), such that the following conditions hold:

1. For all atoms $\varphi \in$ AT($\Sigma^{DOD\text{-}alt}$), if $A \models \varphi$, then $C \models$ IS-PART-OF-DESIGN-PROCESS-RESULTS('φ', 'Pos').
2. For all atoms $\varphi \in$ AT($\Sigma^{DOD\text{-}alt}$), if $A \not\models \varphi$, then $C \models$ IS-PART-OF-DESIGN-PROCESS-RESULTS('φ', 'Neg').
3. There is no other non-tautological sentence $\phi \in$ WFF($\Sigma^{DOD\text{-}control}$) such that $C \models \phi$.

The transition type *overall DOD control downward reflection* models the mapping of overall design strategy information to assumptions for making alterations to design object descriptions.

Definition 4.5.12. (Overall DOD Control Downward Reflection) An *overall DOD control downward reflection* is a pair $\langle C, A\rangle \in \text{IS}(\Sigma^{\text{DOD-control}}) \times \text{IS}(\Sigma^{\text{DOD-alt}})$, such that the following conditions hold:

1. For all P: CONTROL-PROCESS-PLAN, if $C \models$ IS-CURRENT-CONTROL-PROCESS-PLAN(P) then $A \models P$.
2. There is no other non-tautological sentence $\phi \in \text{WFF}(\Sigma^{\text{DOD-alt}})$ such that $A \models \phi$.

Definition 4.5.12 can be further extended such that the design object description alteration state is only updated, which is a more intuitive approach. Note further that for all S: DESIGN-PROCESS-STATE, for all D: DESIGN-STRATEGY, and for all P: CONTROL-PROCESS-PLAN, if $C \models$ IS-CURRENT-CONTROL-PROCESS-STATE(S) holds (i.e., S is the current control process state), $C \models$ INCLUDES-DESIGN-STRATEGY(S, D) holds (i.e., state S includes D as the design strategy), and $C \models$ IS-DEFINED-AS(D, P) holds (i.e., design strategy D is defined in terms of control process plan P), then also $C \models$ IS-CURRENT-CONTROL-PROCESS-PLAN(P) holds (i.e., control process plan P is the current control process plan).

For example, Chapter 12 shows how strategies for generating a design object description are determined. In one example, this results in an overall DOD control state that includes the following information:

```
is-current-control-process-state(DODMState20)
includes-design-strategy(DODMState20, generate-design-from-scratch)
is-defined-as(generate-design-from-scratch,
  is-set-of-criteria-for-DOD-extension-by-retrieval([is-empty]))
```

The result of reflecting this overall DOD control state downwards is a DOD alteration state that includes the following information:

```
is-set-of-criteria-for-DOD-extension-by-retrieval([is-empty])
```

The transition type *RQS alteration transition* models the results of design activities such as the assessment of requirement qualification sets, and the generation, rejection, and selection of proposals for making modifications to the current requirement qualification set.

Definition 4.5.13. (RQS Alteration Transition) An *RQS alteration transition* is a pair $\langle A, A'\rangle \in \text{IS}(\Sigma^{\text{RQS-alt}}) \times \text{IS}(\Sigma^{\text{RQS-alt}})$.

The transition type *RQS alteration upward reflection* models the mapping of epistemic information about the results of making alterations to requirement qualification sets to overall requirement qualification set control information.

Definition 4.5.14. (RQS Alteration Upward Reflection) An *RQS alteration upward reflection* is a pair $\langle A, C\rangle \in \text{IS}(\Sigma^{\text{RQS-alt}}) \times \text{IS}(\Sigma^{\text{RQS-control}})$, such that the following conditions hold:

1. For all atoms $\varphi \in \text{AT}(\Sigma^{\text{RQS-alt}})$, if $A \models \varphi$, then $C \models$ IS-PART-OF-DESIGN-PROCESS-RESULTS('φ', 'Pos').
2. For all atoms $\varphi \in \text{AT}(\Sigma^{\text{RQS-alt}})$, if $A \not\models \varphi$, then $C \models$ IS-PART-OF-DESIGN-PROCESS-RESULTS('φ', 'Neg').
3. There is no other non-tautological sentence $\phi \in \text{WFF}(\Sigma^{\text{RQS-control}})$ such that $C \models \phi$.

The transition type *overall RQS control downward reflection* models the mapping of overall design strategy information to assumptions for making alterations to requirement qualification sets.

Definition 4.5.15. (Overall RQS Control Downward Reflection) An *overall RQS control downward reflection* is a pair $\langle C, A\rangle \in \text{IS}(\Sigma^{\text{RQS-control}}) \times \text{IS}(\Sigma^{\text{RQS-alt}})$, such that the following conditions hold:

1. For all P: CONTROL-PROCESS-PLAN, if $C \models$ IS-CURRENT-CONTROL-PROCESS-PLAN(P) then $A \models P$.
2. There is no other non-tautological sentence $\phi \in \text{WFF}(\Sigma^{\text{RQS-alt}})$ such that $A \models \phi$.

Definition 4.5.15 can be further extended such that the requirement qualification set alteration state is only updated, which is a more intuitive approach.

For example, Chapter 12 shows how strategies for generating a requirement qualification set are determined. In one example, this results in an overall RQS control state that includes the following information:

```
is-current-control-process-state(RQSMState20)
includes-design-strategy(RQSMState20, ignore-initial-soft-client-requirements)
is-defined-as(ignore-initial-soft-client-requirements,
  is-set-of-criteria-for-RQS-reduction-by-modification(
    [is-introduced-at(start-time), has-qualification(all-possible), has-source(client)]))
```

The result of reflecting this overall RQS control state downwards is an RQS alteration state that includes the following information:

is-set-of-criteria-for-RQS-reduction-by-modification(
[is-introduced-at(start-time), has-qualification(all-possible), has-source(client)])

The transition type *overall DOD control transition* models the results of design activities such as the determination of a local strategy for generating a satisfactory design object description, and the evaluation of the design process and its results (as far as design object descriptions are concerned) against the overall design strategy.

Definition 4.5.16. (Overall DOD Control Transition) An *overall DOD control transition* is a pair $\langle C, C' \rangle \in \text{IS}(\Sigma^{\text{DOD-control}}) \times \text{IS}(\Sigma^{\text{DOD-control}})$.

The transition type *overall RQS control transition* models the results of design activities such as the determination of a local strategy for generating a satisfactory requirement qualification set, and the evaluation of the design process and its results (as far as requirement qualification sets are concerned) against the overall design strategy.

Definition 4.5.17. (Overall RQS Control Transition) An *overall RQS control transition* is a pair $\langle C, C' \rangle \in \text{IS}(\Sigma^{\text{RQS-control}}) \times \text{IS}(\Sigma^{\text{RQS-control}})$.

The transition type *overall co-ordination transition* models the results of design activities such as the determination of an overall strategy for the design process, and the evaluation of the design process and its results against the given design process objectives.

Definition 4.5.18. (Overall Co-ordination Transition) An *overall co-ordination transition* is a pair $\langle C, C' \rangle \in \text{IS}(\Sigma^{\text{coord}}) \times \text{IS}(\Sigma^{\text{coord}})$.

Given Definitions 4.5.16 to 4.5.18, an *overall design control transition* is defined as a pair $\langle C, C' \rangle \in \text{IS}(\Sigma^{\text{design-control}}) \times \text{IS}(\Sigma^{\text{design-control}})$.

A *design trace* is a record of the history of a specific design process. Design traces are useful not only for the purpose of inspection, benchmarking, learning, or teaching, but also for re-use (of specific lines of reasoning or design results) in a new design process.

Definition 4.5.19. (Design Trace) Let Σ^{design} be the order-sorted signature $\Sigma^{\text{DO}} \cup \Sigma^{\text{basic-eval}} \cup \Sigma^{\text{RQ}} \cup \Sigma^{\text{DOD-alt}} \cup \Sigma^{\text{RQS-alt}} \cup \Sigma^{\text{design-control}}$. A *design trace M* is a partial temporal Σ^{design}-model, such that the following conditions hold:

1. The transition $\langle M(1), M(2) \rangle$ is an overall co-ordination transition.
2. For each integer i greater than 1, $\langle M(i\text{-}1), M(i) \rangle$ is a transition of a type defined in this section.

A *temporal design model* describes the possible (intended) behaviour of a design system: from every possible initial design state, design traces from the temporal design model can be generated by following the different possible transitions from that design state.

Definition 4.5.20. (Temporal Design Model) For a specific design system *DS*, a set *S* of design traces is called a *temporal design model* of *DS* if the design trace of each design process that can be executed with *DS* is a member of *S*. The smallest temporal design model of *DS* (in terms of the size of the set) is referred to as TRACES_{DS}.

All design traces actually generated in practice by a design system *DS* together form the set $\text{BEHAVIOUR-MOD}_{DS}$, which is a subset of TRACES_{DS}. An example of dynamic properties of a design system is given at the end of this section.

A *design pattern* is a recurring sequence within design traces of design processes. When frequent re-use is expected or promoted, it is worthwhile to investigate design patterns. A design pattern is especially powerful in combination with *design rationale*, which is a set of justifications of the design decisions within a design trace or design pattern.

A *design history* is a set of one or more design traces of design processes (in the past), together with their design rationale. In most design systems, only part of the design history is (or, needs to be) represented; for instance, for the type of elevator configuration processes described in Chapter 10, it was sufficient to remember the previous state of the elevator configuration process when revising an elevator configuration.

Based on the definition of a design trace, it is now possible to define a *design solution* (Definition 4.5.24) as the combination of three aspects: a satisfactory requirement qualification set (Definition 4.5.21), a satisfactory design object description (Definition 4.5.22), and a satisfactory design process (Definition 4.5.23).

Definition 4.5.21. (Requirement Qualification Set Solution) Let Σ^{RQ} be a design requirements vocabulary. At the current time point in a design process, a requirement qualification set S_1 over Σ^{RQ} is a *solution* to a requirement qualification set S_2 over Σ^{RQ} if:

1. S_1 can be fulfilled;
2. S_1 is consistent, unambiguous, precise, and complete;
3. so far, the client commits to S_1 as an acceptable substitute for S_2.

These conditions can be formalised as follows:

$$\begin{aligned}
&\mathbf{C}\text{CAN-BE-FULFILLED}(S_1) \wedge \\
&\mathbf{C}\neg\text{IS-INCONSISTENT}(S_1) \wedge \mathbf{C}\neg\text{IS-AMBIGUOUS}(S_1) \wedge \\
&\mathbf{C}\neg\text{IS-IMPRECISE}(S_1) \wedge \mathbf{C}\neg\text{IS-INCOMPLETE}(S_1) \wedge \\
&\neg\mathbf{P}\neg\text{IS-ACCEPTABLE-SUBSTITUTE-FOR}(S_1, S_2) \wedge \\
&\mathbf{C}\text{IS-ACCEPTABLE-SUBSTITUTE-FOR}(S_1, S_2)
\end{aligned}$$

Note that, as a special case, a well defined requirement qualification set that can be fulfilled and that is acceptable to the client is a requirement qualification set solution to itself.

Definition 4.5.22. (Design Object Description Solution) Let Σ^{DO} be a domain object vocabulary, Φ a domain object theory over Σ^{DO}, and Σ^{RQ} a design requirements vocabulary. At the current time point in a design process, a design object description D over Σ^{DO} is a *solution* (under Φ) to a requirement qualification set S over Σ^{RQ} if:

1. D is consistent;
2. the BASIS-reduct of D fulfils S (under Φ).

These conditions can be formalised as follows:

$$\mathbf{C}\text{IS-CONSISTENT-WITH}(D, D) \wedge$$
$$\mathbf{C}\text{IS-BASIS-REDUCT-OF}(D', D) \wedge \mathbf{C}\text{FULFILS}(D', S)$$

Note that, as a special case, an empty design object description (i.e., one that does not include any domain object information) is a solution to an empty requirement qualification set (i.e., one that does not include any design requirement information).

Definition 4.5.23. (Design Process Solution) At the current time point in a design process, the design process is *satisfactory* (with respect to the given design process objectives) if for every design process objective O to be satisfied, it is the case that O is satisfied.

This condition can be formalised for a specific design process objective O as follows:

$$\mathbf{C}\text{IS-TO-BE-SATISFIED}(O) \Rightarrow \mathbf{C}\text{IS-SATISFIED}(O)$$

Note that, as a special case, a design process that need not satisfy any design process objective is by definition satisfactory.

Definition 4.5.24. (Design Solution) Let Σ^{DO} be a domain object vocabulary, Φ a domain object theory over Σ^{DO}, D a design object description over Σ^{DO}, Σ^{RQ} a design requirements vocabulary, and S_1 and S_2 two requirement qualification sets over Σ^{RQ}. Assume that the design process of concern has started with S_1 as the initial requirement qualification set. Then at the current time point, the design process has reached a *design solution* $\langle S_1, S_2, D\rangle$ if:

1. the requirement qualification set S_2 is a solution to S_1;
2. the design object description D is a solution to S_2;
3. the design process is satisfactory.

Using these solution definitions, it is possible, for example, to state that a specific design system *DS* never produces two consistent design object description solutions to a given requirement qualification set. This statement can be formalised as follows: for every design trace $M \in \text{BEHAVIOUR-MOD}_{DS}$ requirement qualification set *S*, and design object descriptions D_1 and D_2, *M* is a model of

$$(\mathbf{C}\text{IS-DOD-SOLUTION-TO}(D_1, S) \wedge \mathbf{C}\text{IS-DOD-SOLUTION-TO}(D_2, S)) \Rightarrow \mathbf{C}\neg\text{IS-CONSISTENT-WITH}(D_1, D_2)$$

4.6 Discussion

This chapter has presented the dynamic aspects of design: concepts and their logical relationships that apply to each possible execution path of a design process. For a knowledge-level analysis of these dynamic aspects, temporal logic has been used to express design process behaviour and design process objectives. Furthermore, partial temporal models have been used as interpretations of design process behaviour.

One of the contributions of the work presented in this chapter is a formal definition of the relation between reasoning at different levels of reflection within a design process. A logical theory of both the static aspects and the dynamic aspects of design as outlined in this chapter and the previous chapter can play a useful role in establishing and proving properties of design support systems, such as consistency, correctness, and completeness of their behaviour. For example, proof techniques or model-checking techniques in temporal logic can be used to derive whether a particular design support system is able to generate a given design object description, based on a given requirement qualification set. Deployment of such techniques paves the road to the development of automated tools that support the verification of behavioural properties. For example, the results of the research by Treur and Willems and by Leemans, Treur and Willems on the verification of single-component knowledge-based systems can be used to this end [Treur and Willems, 1994; Leemans, Treur and Willems, 2002]. Also the results of the research by Cornelissen, Jonker and Treur and by Engelfriet, Jonker and Treur on compositional verification are of interest [Cornelissen, Jonker and Treur, 2002; Engelfriet, Jonker and Treur, 2002].

Another contribution of the work presented in this chapter is the definition of a design solution, which explicitly takes the temporal aspects of a design process into account. The definition acknowledges that in practice, the initial requirement qualification set with which a design process starts need not be the same requirement qualification set as the one with which the design process stops. Furthermore, the definition acknowledges that the design process itself may be subject to process requirements (i.e., design process objectives from the client or other parties), which need to be satisfied. The notion of design solution presented in this chapter may act as a starting point for further research into more refined notions.

A limitation of the work presented in this chapter is that it provides no particular insight into dynamic aspects such as distributed design, creativity in design, situatedness in design, and learning in design. This thesis abstracts from the involvement of multiple (co-operative) design agents in a design process, such as the designer, the client, the user, and the legislator. Furthermore, this chapter pays little attention to the situated acquisition of design knowledge, which is a common denominator of creativity, situatedness, and learning in design—further research on these subjects is required.

Chapter 5

Modelling Systems in DESIRE

The component-based multi-agent design method DESIRE supports structured design of autonomous, interactive, component-based systems. DESIRE views individual agents, their tasks and multi-agent systems as component-based structures and supports the evolutionary development of such structures (e.g., design support systems). This chapter describes the main principles underlying DESIRE and how to model component-based systems in DESIRE.

Account. *This chapter is largely based on the work of the Department of Artificial Intelligence of the Vrije Universiteit Amsterdam. Publications on modelling component-based systems in DESIRE by Brazier, Jonker and Treur have been used and adapted by kind permission of the authors [Brazier, Jonker and Treur, 1998; Brazier, Jonker and Treur, 2000].*

The component-based multi-agent design method DESIRE supports structured design of autonomous, interactive, component-based systems. This method supports explicit models of *intra-agent functionality* (i.e., knowledge, reasoning and acting abilities required to perform the tasks for which an agent is responsible) as well as *inter-agent functionality* (i.e., knowledge, reasoning and acting abilities required to perform and guide agent co-ordination, co-operation and other forms of social interaction).

DESIRE (an acronym of "DEsign and Specification of Interacting REasoning components") uses compositionality (i.e., a component-based perspective) and evolutionary development as main guiding principles. As such, DESIRE has a unique place among the current system design and development methods. In this chapter, its main features are summarised.

This chapter is organised as follows. Section 5.1 describes the main principles of compositional design of complex reasoning systems. Section 5.2 describes the design and reuse of generic models as part of the evolutionary development of component-based systems.

5.1 Principles of Compositional Design of Reasoning Systems

The two main guiding principles underlying DESIRE are compositionality and evolutionary development. *Compositionality* means that the individual agents, their tasks and multi-agent systems are viewed as component-based structures: all functionality is modelled in terms of interacting components. Complex distributed processes are seen as the result of tasks performed by agents in interaction with their environment. Hence, these processes can be understood by studying the tasks and the way these tasks are composed into more complex tasks, individual agents and multi-agent systems.

The second main guiding principle of DESIRE is its support of *evolutionary development* of reasoning systems. DESIRE does not assume a fixed sequence of designing such systems: instead, it is assumed that, depending on the specific situation, it may be that different types of knowledge are available at different points during system design. Therefore, DESIRE is based on the view that a system development method should support an iterative approach.

The following sub-sections of this section describe how these principles are incorporated into DESIRE.

5.1.1 The process of designing a component-based system

The design of a component-based system is an iterative process, which aims at the identification of the parties involved (i.e., human agents, software agents, external worlds), and the processes, in addition to the types of knowledge needed. During a component-based system development process, DESIRE distinguishes a problem description, a conceptual design, a detailed design, an operational design and design rationale (see Figure 5.1).

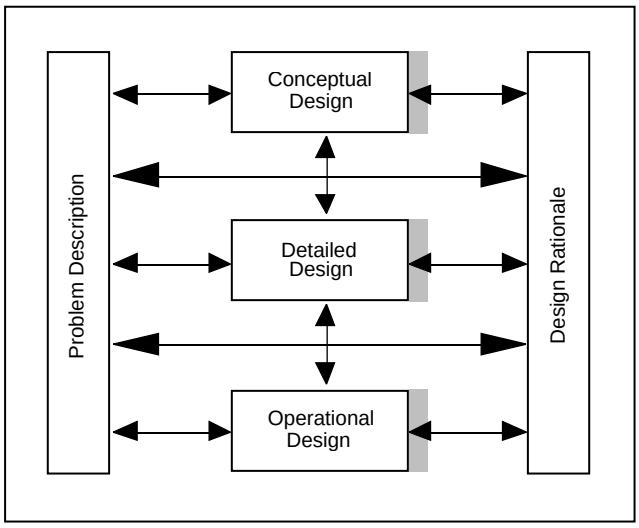


FIGURE 5.1. Problem description, levels of design and design rationale.

The *problem description* includes the *requirements* imposed on the design. The *rationale* specifies the choices made during design at each of the design levels (i.e., conceptual design, detailed design and operational design), and information about the context in which these choices were made. The relationship between the design levels is well defined and structure preserving. The *conceptual design* includes conceptual models for each individual agent and its tasks, the external world, the interaction between agents, and the interaction between agents and the external world. The *detailed design* of a component-based system, based on the conceptual design, specifies all aspects of a system's knowledge and behaviour, which forms an adequate basis for the *operational design.* Prototype implementations are automatically generated from the detailed design.

There is no fixed sequence of design: depending on the specific situation, different types of knowledge are available at different points during system design. Conceptual descriptions of specific processes and knowledge are often first attained. Further explication of these descriptions results in detailed design descriptions, usually in iteration with conceptual design. During the design of these models, partial prototype implementations may be used to analyse or verify the resulting behaviour. On the basis of assessment of these partial prototypes, new designs and prototypes are generated and examined, and so forth and so on. This approach to *evolutionary development* is characteristic to the development of systems in DESIRE.

The end result is a component-based system design, specified by the system designer at the level of detailed design. In addition, important assumptions and design decisions are specified in the design rationale, stating alternative design options together with argumentation. On the basis of verification during the design process, properties of a model under development can be documented with the related assumptions. The assumptions define the limiting conditions under which the model will exhibit specific behaviour.

5.1.2 Compositionality of processes and knowledge

Compositionality is a general principle that refers to structuring a system from a component-based perspective. The design method DESIRE structures both processes and knowledge in a compositional manner.

In a component-based system, complex processes are component-based structures in which a number of other, more detailed processes happen. During the design of a component-based system according to the method DESIRE, different levels of process abstraction are identified. Processes at each of these levels (except those at the lowest level) are modelled as (process) *components* that are composed of components at the adjacent lower level.

The *ontology* that expresses the knowledge needed to reason about a specific domain of application may also be seen as a single (knowledge) component. This *knowledge structure* can be composed of a number of more specific knowledge structures which, in turn, may again be composed of other, even more specific knowledge structures.

As is shown in Figure 5.2, *compositionality of processes* and *compositionality of knowledge* are two separate, orthogonal dimensions of component-based systems. The compositional knowledge structures are associated to the compositional process structures.

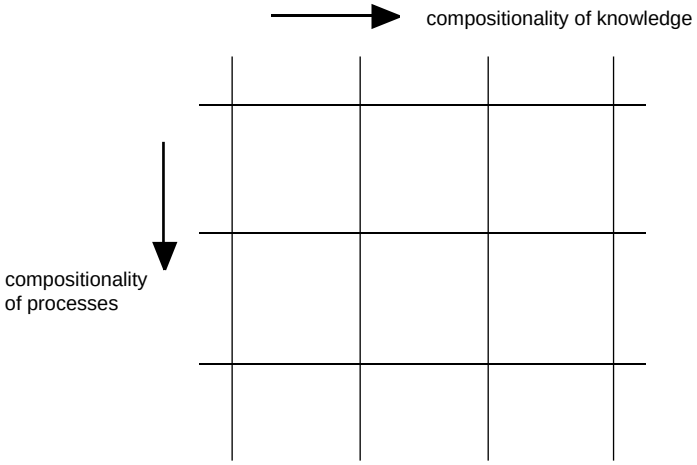


FIGURE 5.2. Compositionality of processes and compositionality of knowledge.

Compositionality is a means to achieve *information and process hiding* within a model: by defining processes and knowledge at different levels of abstraction, unnecessary details can be hidden. Compositionality also makes it possible to *integrate* different types of components in one agent. Components and groups of components can easily be included in new designs, supporting *reuse* of components at all levels of design.

5.1.3 Problem description

A problem description is formed by acquisition of requirements to be imposed on the system to be developed. These requirements become part of the initial problem description, but they may evolve during the development of a system. Which techniques are used to acquire a problem description is not pre-defined. Techniques vary in their applicability, depending on, for example, the domain of application, the task, and the type of knowledge on which the system developer wishes to focus.

5.1.4 Conceptual design and detailed design

A conceptual design and a detailed design each consist of specifications of the following three types:

- process composition,
- knowledge composition,
- the relation between process composition and knowledge composition.

These three types of specifications are discussed in more detail below.

5.1.4.1 Process composition

Process composition identifies the relevant processes at different levels of (process) abstraction, and describes how a process can be defined in terms of lower level processes. Depending on the context in which a system is to be designed, two different views can be assumed: a task perspective and a multi-agent perspective. The *task perspective* refers to the view according to which the processes needed to perform an overall task are designed first. These processes (or sub-tasks) are then *delegated* to appropriate agents and the external world, after which these agents and the external world are designed. The *multi-agent perspective* refers to the view according to which agents and an external world are designed first and then the processes within each agent and within the external world.

5.1.4.1.1 Identification of processes at different levels of abstraction

Processes can be described at different levels of abstraction: processes for a multi-agent system, processes within individual agents and the external world, and processes within task-related components of individual agents.

Modelling a process. The identified processes are modelled as *components*. For each process the *types of information* used as input and produced as output are identified and modelled as *input and output interfaces* of the component.

Table 5.1 presents a specification of the interface information types of an agent. It shows that an agent has two input information types (incoming communication and observation result info) and three output information types (outgoing communication, observation info and action info).

TABLE 5.1. Specification of interface information types of an agent.

Process	*Input information types*	*Output information types*
agent	incoming communication, observation result info	outgoing communication, observation info, action info

Modelling process abstraction levels. The identified levels of process abstraction are modelled as *abstraction/specialisation relations* between components at adjacent levels of process abstraction: components may be *composed* of other components or they may be *primitive*. Primitive components may be reasoning components (for example, based on a knowledge base), or, alternatively, components capable of performing tasks such as calculation, information retrieval, optimisation, et cetera.

Figure 5.3 shows the abstraction relations between processes at the two highest process abstraction levels of an agent. For example, the figure shows that an agent has agent interaction management as a sub-process.

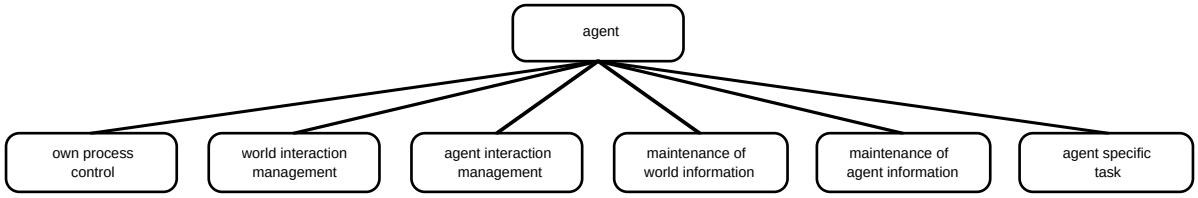


FIGURE 5.3. Processes at the two highest process abstraction levels of an agent.

The identification of processes at different abstraction levels results in a specification of components that can be used as building blocks, and in a specification of the sub-component relation, defining which component is a sub-component of which other component. The distinction of different process abstraction levels results in process hiding.

5.1.4.1.2 Composition of processes

The way in which processes at one level of abstraction in a system are composed of processes at the adjacent lower abstraction level in the same system is called *composition*. The composition of processes is described by the component/sub-component relations and by the possibilities for *information exchange* between processes (assuming a *static view* on the composition) and the *task control knowledge* used when appropriate to control processes and information exchange (assuming a *dynamic view* on the composition).

Information exchange. A specification of information exchange defines which types of information can be transferred between components and the *information links* by which this can be achieved. Within each of the components, *private* information links are defined to transfer information from one sub-component to another. In addition, *mediating* links are defined to transfer information as follows: (a) from the input interfaces of the encompassing component to the input interfaces of the sub-components, (b) from the output interfaces of the sub-components to the output interface of the encompassing component, and (c) directly between the input interface and the output interface of the encompassing component.

Figure 5.4 presents information exchange between processes at the highest process abstraction levels within an agent. For example, it shows that the mediating link communicated info transfers information from the input interface of the component agent to the input interface of the component agent interaction management, and that the private link communicated agent info transfers information from the output interface of the component agent interaction management to the input interface of the component maintenance of agent information.

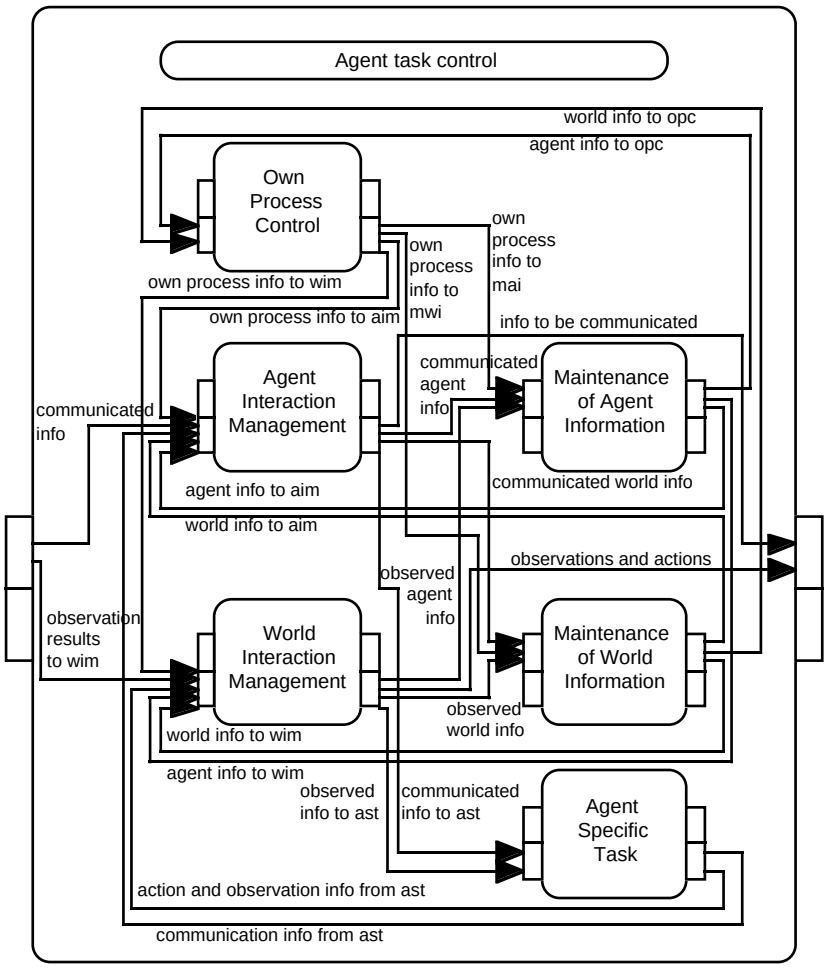


FIGURE 5.4. Information exchange between processes at the highest process abstraction levels of an agent.

Task control. Depending on the situation, processes within a component-based system may be performed sequentially or in parallel. Some processes may be continually performed (e.g., reacting immediately to new input) and some only in specific situations. The same holds for information exchange, which may take place continually or only in specific situations. Task control determines when and how processes are to be performed and evaluated: evaluation of the results of components (success or failure) provides a means to further guide processing.

The design of a component-based system specifies which components are to be activated sequentially and which ones are *awake*, meaning that they are supposed to process new input as soon as it arrives. *Task control knowledge* specifies which components and information links are awake or when they are to be activated. Goals of a process are defined by *task control foci* together with the *extent* to which they are to be pursued. Evaluation of the success or failure of a process's performance is specified by *evaluation criteria* together with an extent.

5.1.4.2 Knowledge composition

Knowledge composition identifies knowledge structures at different levels of (knowledge) abstraction and describes how a knowledge structure is composed of lower-level knowledge structures. The knowledge abstraction levels may correspond to the process abstraction levels, but this is not often the case. Usually, the matrix depicted in Figure 5.2 shows an *m* to *n* correspondence between processes and knowledge structures, with *m* and *n* larger than 1.

5.1.4.2.1 Identification of knowledge structures at different abstraction levels

The two main knowledge structures used as building blocks to model knowledge are *information types* and *knowledge bases*. These knowledge structures can be identified and each described at different levels of abstraction. As a result, at the higher abstraction levels the details are hidden. The resulting levels of knowledge abstraction can be distinguished for both information types and knowledge bases.

Information types. An information type defines an *ontology* (i.e., a lexicon or vocabulary) to describe objects, their sorts, and the relations and functions that can be defined on these objects. Information types are defined as signatures for order-sorted predicate logic (with sets of names for sorts, objects, functions and relations). Information types can be specified in a graphical form or in a formal textual form.

Figure 5.5 shows the structure of the information type incoming communication present in the input interface of an agent; it shows the specification of a ternary relation communicated-by with three arguments of the sorts INFO-ELEMENT, SIGN and AGENT, respectively.

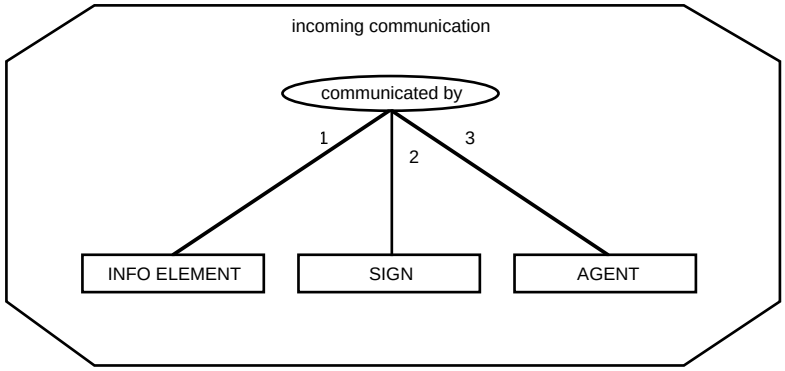


FIGURE 5.5. Structure of the information type incoming communication.

Knowledge bases. Knowledge bases use ontologies defined in information types. The relation between information types and knowledge bases defines which information types are used in which knowledge bases. In detailed design, knowledge bases define relationships between the concepts specified in the information types.

For example, knowledge from the knowledge base communicated info extraction kb, of which the purpose is to abstract the information communicated to an agent from the specific agent communicating this information, can be written in semi-formal form as follows:

if information element E is communicated (with sign S) by agent A,
then information element E (with sign S) is new world information.

This knowledge base may be used within the component agent interaction management to identify the communicated information that needs to be maintained. In concise formal form, the above knowledge is specified as follows:

if communicated-by(E: INFO-ELEMENT, S: SIGN, A: AGENT)
then new-world-information(E: INFO-ELEMENT, S: SIGN);

5.1.4.2.2 Composition of knowledge structures

Information types can be composed of more specific information types, following the principle of compositionality explained above. Similarly, knowledge bases can be composed of more specific knowledge bases. The component-based structure is based on the different levels of knowledge abstraction distinguished, and results in information and knowledge hiding. Figure 5.6 shows the composition of the information action info, which is present in the output interface of an agent.

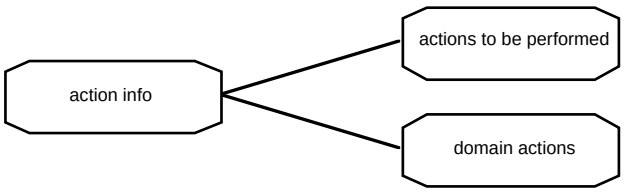


FIGURE 5.6. Composition of the information type action info.

5.1.4.3 Relation between process composition and knowledge composition

Each process in a process composition uses knowledge structures. The relation between the process composition and the knowledge composition of a component-based system defines which knowledge structures (information types and knowledge bases) are used for which processes. The rows within the matrix depicted in Figure 5.2 specify this relation.

Table 5.2 presents the relation between some knowledge bases and processes within an agent. It shows, for example, that the knowledge base communicated info extraction kb is used in the process agent interaction management.

TABLE 5.2. Relation between knowledge bases and processes within an agent.

Knowledge base	*Process using the knowledge base*
observation result extraction kb	world interaction management
communicated info extraction kb	agent interaction management

5.1.5 Design rationale

The *design rationale* describes the relevant properties of the designed system in relation to the design requirements identified in the problem description. The design rationale further provides documentation for verification of the design, including assumptions on situations under which the required properties hold. Important design decisions are made explicit, together with some of the alternative choices that could have been made and the arguments in favour of and against the different choices. At the operational level, the design rationale includes decisions based on operational considerations, such as the choice to implement a parallel process on one or more machines, depending on the available capacity.

5.2 Design and Reuse of a Generic Model

During the design of a specific agent or task model, often a number of generic processes can be identified. This sub-section describes the design of a generic model, which includes specifications of such generic processes, abstracting from the specific domains of application. The reason to design a generic model is that it can be (re) used in the design of a large variety of agents and/or agent-specific tasks. In the development of a component-based system, reuse of generic models may enhance the design process as well as the result, and save a lot of time.

5.2.1 Generic models and reuse

Instead of designing each and every model of an agent or an agent-specific task from scratch, an existing generic model can be used in which always-occurring processes and knowledge structures are explicitly modelled. The use of a generic model structures the design process: the acquisition of a conceptual model for a specific application can be based on the generic structures present in the generic model. The model is generic in two senses: it is generic with respect to the *processes* as well as the *knowledge* involved in a specific application.

Genericity with respect to processes refers to levels of process abstraction: a generic model abstracts from more detailed processes at lower levels. A more specific model, which distinguishes more specific processes at a lower level of process abstraction, is obtained from a generic model by refinement of the generic model; this type of refinement is called *specialisation*.

Genericity with respect to knowledge refers to levels of knowledge abstraction: a generic model abstracts from more detailed knowledge structures. A more specific model, which distinguishes more specific knowledge structures for specific domains of application, is obtained from a generic model by refinement of the generic model; this type of refinement is called *instantiation*.

A generic model can be reused for a wide variety of more specific models by means of refinement and composition. Reuse as such reduces the time, expertise and effort needed to design and maintain system designs. Which components, links and knowledge structures from a generic model are applicable in a given situation depends on the tasks that an agent needs to be able to perform. Whether a component from a generic model can be used immediately, or whether instantiation, modification and/or specialisation is required, depends on the desired functionality of an agent.

It may be that a specific task or agent involves processes and knowledge structures to which more than one generic model applies. A combination of existing generic models can be used for specialisation and instantiation of a model. Which generic models are suited depends on the problem description: existing components and knowledge structures are examined, rejected, modified, specialised or instantiated in the context of the problem at hand.

The component-based design method DESIRE provides a means to specify a generic (agent) model: a model in which generic components and their interactions are distinguished. That is, by means of DESIRE both generic models and specific models of tasks and agents can be developed.

5.2.2 Designing and reusing a generic model: the process

This sub-section summarises the process of designing and reusing a generic (agent) model.

5.2.2.1 Designing a generic model

A generic (agent) model is usually not invented from scratch, but the result of a (possibly long) process of empirically studying practical applications, investigation of related research and many (partially successful) design efforts. Conceptual analysis of process characteristics is the main rationale for the components distinguished in a generic (agent) model, in order to abstract from the details from specific domains of applications and process methods.

For example, the generic design model GDM presented in this thesis has not been designed from scratch. Its components have been distinguished in design processes in different application domains, such as those presented in Part IV and Part V, which were an important input for the process of designing GDM in more detail. Generic structures were extracted from these example models and combined, leaving out domain-specific elements. For example, the components and knowledge structures related to design object description modification are based on those made for elevator configuration design (see Chapter 10).

Often a trade-off has to be made on the amount of support that a generic model will provide. On the one hand, the more structures are included, the more support is given when reusing the generic model. On the other hand, the richer the structure of a generic model is, the more restrictive its scope of application may be. In GDM, for example, the component DOD assessment has not been refined, in order to leave open the use of, for example, constraint-based approaches to design. Since GDM has been designed to be a widely applicable model, the choice has been made to leave out more specific commitments to assessment processes.

5.2.2.2 Reusing a generic model

In general, a generic (agent) model can be reused in the following way:

- most parts of the generic model (i.e., components, information types, information links, task control, and knowledge bases) can be reused as is,
- unused parts are modified, remain empty or are removed,
- necessary additional knowledge structures or information links are added,
- necessary additional components are added or existing components are modified.

Reusing the generic design model GDM to model specific design applications and design themes (of which the results are presented in Parts IV and V) has shown almost all of these characteristics.

Chapter 6

GDM: a Generic Design Model

GDM is a generic design model that provides a blueprint of the component-based structure of design processes. GDM is a useful starting point for modelling design processes in different domains and for the development of many types of design support systems. This model is based on our logical theory of design and experiences in different applications. GDM models three levels of process abstraction; in this chapter, the highest two levels are described.

Publications. *The development of GDM and its applications has been addressed in a number of papers. An initial version of GDM has been presented at the AID '94 Conference in Lausanne, Switzerland [Brazier, Langen, Ruttkay and Treur, 1994]. Refined and improved versions of GDM have been used to model conflict management in design [Brazier, Langen and Treur, 1995*b*], re-design of knowledge-based systems [Brazier, Langen, Treur and Wijngaards, 1996], elevator configuration design [Brazier, Langen, Treur, Wijngaards and Willems, 1996], design rationale [Brazier, Langen and Treur, 1997], and strategic design knowledge [Brazier, Langen and Treur, 1998*a*; Brazier, Langen and Treur, 1998*b*].*

Design support systems are computer systems that support human designers during design. Such systems are capable of accessing, processing and storing huge amounts of design data and most often offer a means to visualise designs. In the last decade, Computer Aided Design systems (CAD systems) have become an integral part of professional design. Electronic engineers, architects, civil engineers, car designers, aircraft engineers, and ship builders all use CAD systems to produce designs. To date, these systems provide tools for two-dimensional and three-dimensional drafting, drawing, constraint checking, graphical presentation and simulation. The development and employment of faster (personal) computer technology and graphical user interfaces have given a tremendous push to the usability of such systems.

Although current CAD systems are quite advanced, there is still a lot of work to do as far as the support of the human designer is concerned. Generally, CAD systems operate without knowledge of the semantics of the design objects involved, the requirements that can be imposed on design objects, and the designers' methods to bring design processes to a successful end. Stated differently, most CAD systems do not know what the user is doing and why. For the development of more advanced design support systems that provide human designers with real support for reasoning about various aspects of design, it is essential to have an integral model of the composition of design processes, application domains, and design methods.

In practice, a design process involves not only the construction of a description of an artefact, but also the determination of (additional or substitute) requirements of the artefact, and the strategic co-ordination of these activities. Based on this notion, a generic model of design processes has been developed, called GDM. An initial version of this generic model has been presented at the Artificial Intelligence in Design '94 Conference [Brazier, Langen, Ruttkay and Treur, 1994] and has undergone several changes since.

GDM is a blueprint of the generic features of design processes. GDM models the essential types of information and knowledge that play a role within a design process, irrespective of application domains and design methods. By using GDM as a basis for modelling a specific type of design process, a modeller needs to focus only on the specifics of the application domain and the design methods that may be used. Our claim is that this approach is a powerful means to develop design support systems in a well-founded manner.

GDM is based on our logical theory of design (Chapters 3 and 4) and various experiences in applied research. See, for example, the design of routes for international blank payments [Geelen and Kowalczyk, 1992], the allocation of rooms to staff [Geelen, Ruttkay and Treur, 1992] and the design of packages of environmental measures [Brazier, Treur and Wijngaards, 1996*b*].

In GDM, three levels of process abstraction are distinguished (from a task perspective). This chapter describes the two highest levels of a design process, abstracting from possible actors in a design process such as the client, the designer and the design system. A further conceptual elaboration of GDM can be found in Chapters 7 and 8, which, together, describe the third abstraction level of a design process.

This chapter is organised as follows. Section 6.1 presents the process composition of a design process, Section 6.2 the knowledge composition of a design process, and Section 6.3 the relation between process composition and knowledge composition of a design process. Finally, Section 6.4 explains how the model of the two highest levels of a design process that GDM provides can be used in the analysis of practical design processes and the development of design support systems.

GDM has been developed with the component-based development method DESIRE. For a detailed textual specification of GDM expressed in the language of DESIRE, the interested reader is referred to Appendix B, which can be read alongside Chapters 6 to 8.

6.1 Process Composition

This section describes processes identified in design at different levels of abstraction, and the composition of these processes.

6.1.1 Processes at different abstraction levels

This section describes processes involved in design and the different levels of abstraction at which these processes play a role.

6.1.1.1 Processes

The following processes are involved at the two highest levels of process abstraction of a design process:

- design (*design*),
- design process co-ordination (*DPC*),
- requirement qualification set manipulation (*RQSM*), and
- design object description manipulation (*DODM*).

In the following, these four processes are explained, together with the types of input information they use and the types of output information they produce.

Design

A design process, as a whole, generates a design object description that fulfils a specific set of design requirements. During the process, not only design object descriptions, but also design requirements, may be generated and modified. In some application domains, the requirements given as input may be rather abstract and may need to be replaced by concrete requirements expressing measurable criteria. In other domains, design requirements may have to be revised because of new ideas that emerge during the design process. Furthermore, to bring the design process to a successful end in view of given design process objectives, some form of co-ordination may be needed: an overall design strategy that directs and constrains the generation and modification of both design requirement sets and design object descriptions. This complex process is modelled in GDM by the component design.

This notion of design process covers a broad range of application domains. On one hand, there are domains where design requirements express concrete, measurable criteria that are taken as they are and not modified (as encountered often in mechanical engineering). On the other hand, there are domains where design requirements may express relatively vague, abstract needs and desires and where the initially generated design object descriptions are merely sketches (as encountered often in architecture and industrial design).

Figure 6.1 shows (on the left-hand side) the types of information that a design process uses as input and (on the right-hand side) the types of information that it produces as output.

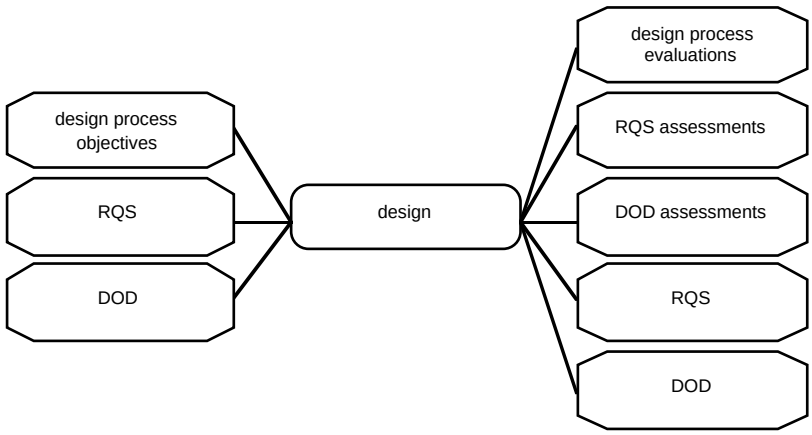


FIGURE 6.1. Input and output of a design process.
(RQS = requirement qualification set, DOD = design object description.)

The following example is used to explain the types of input information used by a design process.

Example 6.1.
"The artefact to be designed is a bicycle. The design task must be completed within 160 hours. The requirements for the bicycle include that it must be suited for riding in mountains, low-priced and safe to be used by young children. If these latter two requirements are irreconcilable, safety is preferred over a low price; the first requirement, though, cannot be negotiated. As a starting point, an earlier drawing of a bicycle is available. This drawing includes the information that this particular bicycle has two wheels, a frame, a saddle, a steer and a six-gear motion-transfer system, that the frame contains shock absorbers, and that the wheels are made of carbon."

Design process objectives. A design process has to take aspects into account such as goals, customer(s), users, deliverables, milestones, budget and resources. (A design process is often a project itself or an activity within a project.) *Design process objectives* address these overall aspects of a design process; they are modelled in GDM by the information type design process objectives. (For the purpose of reference, design process objectives may have names that are unique within the context of a design process.) Two types of design objectives are distinguished: process objectives, and qualified process objectives.

A *process objective* is an objective that can be imposed on a design process (e.g., to meet a specific deadline). Definitions of process objectives are modelled in GDM by the information type process objective definitions. A *qualified process objective* is the combination of a qualification (i.e., an absolute or relative measure of importance) and one or more process objectives, stating which process objectives must be satisfied and which ones may be violated, if unavoidable. Definitions of qualified process objectives are modelled in GDM by the information type qualified process objective definitions. See Table 6.1 for the design process objectives in Example 6.1.

TABLE 6.1. Design process objectives in Example 6.1.

Kind	*Content*
Design process objective	The design task must be completed within 160 hours. [DPO1]

RQS. In a design process, the goal is to construct a description of a design object that satisfies specific *design requirements* concerning the behaviour, function, form and/or structure of this object. (For the purpose of reference, design requirements may have names that are unique within the context of a design process.) A *requirement qualification set* is a set of such design requirements. A characteristic feature of a design process is that it has a (partial) requirement qualification set as input. In GDM, the information type RQS is used to model requirement qualification sets. (For the purpose of reference, requirement qualification sets have names that are unique within the context of a design process.) Two types of design requirements are distinguished: requirements, and qualified requirements.

A *requirement* is a specific condition on a design object regarding its behaviour, function, form or structure. One requirement may denote a concrete, measurable criterion: the answer to whether or not it is satisfied by a specific design object description is a simple yes or no. Another requirement may denote an abstract, ambiguous need or desire (e.g., put forward by a customer), which is to be replaced during a design process by one or more concrete, unambiguous requirements. Definitions of requirements are modelled in GDM by the information type requirement definitions.

A *constraint* is a requirement that prescribes the acceptable values of a specific attribute of a design object (e.g., colour, weight or size), either by enumerating these values or by declaring boundary values. A constraint is usually imposed not by choice (i.e., not by a party involved in the design such as the customer or the designer), but by obligation. For example, fire prevention regulations pose constraints on the design of a house; an architect must ensure that they are met, otherwise local government will not give permission to build the house. In GDM, constraints are modelled in the same way as requirements.

A *qualified requirement* is a specific combination of a qualification (i.e., an absolute or relative measure of importance) and one or more requirements, stating which requirements must be satisfied and which ones may be violated, if unavoidable. Definitions of qualified requirements are modelled in GDM by the information type qualified requirement definitions. See Table 6.2 for the requirement qualification set (named RQS1) in Example 6.1.

TABLE 6.2. Requirement qualification set in Example 6.1.

Kind	*Content*
Requirement	The bicycle must be suited for riding in mountains. [R1]
Requirement	The bicycle must be low-priced. [R2]
Requirement	The bicycle must be safe to be used by young children. [R3]
Qualified requirement	It is necessary to satisfy R1. [QR1]
Qualified requirement	If irreconcilable, satisfying R3 is preferred over satisfying R2. [QR2]
RQS	RQS1 includes as design requirement information the requirements R1, R2 and R3 as well as the qualified requirements QR1 and QR2.

DOD. A *design object description* provides a (partial) description of a design object (and possibly other domain objects) in terms of behaviour, function, form and/or structure. Each design object description includes a number of domain object information elements. A characteristic feature of re-design processes is the role of existing design object descriptions as input; an existing design object description may have to be modified, for example, because of flaws detected in the original design or because of new functionality required of an existing design object. In GDM, the information type DOD is used to model design object descriptions. (For the purpose of reference, design object descriptions have names that are unique within the context of a design process.) See Table 6.3 for the design object description (named DOD1) in Example 6.1.

TABLE 6.3. Design object description in Example 6.1.

Kind	*Content*
DOD	DOD1 includes the domain object information that (a) the bicycle has two wheels, a frame, a saddle, a steer and a six-gear motion-transfer system, (b) the frame contains shock absorbers and (c) the wheels are made of carbon.

The following example, extending Example 6.1, is used to explain the types of output information produced by a design process.

Example 6.2.
"It is known that the design task has been completed successfully in 153 hours, which is within the limit of 160 hours. A new set of design requirements has been generated, based on the original set. For example, there is a new requirement that the bicycle's price must be less than 400 euro. Also more detailed safety requirements have been introduced, such as the requirement that the bicycle must have light reflectors. A new design has been generated that fulfils the new set of design requirements; for example, it satisfies the requirement that the bicycle be suited for riding in mountains. The new design includes the domain object information that the bicycle costs 350 euro, that the bicycle has light reflectors, and that the wheels are made of stainless steel."

Design process evaluations. A design process may succeed or fail to meet design process objectives. *Design process evaluations* provide evaluations of the results of a design process compared to the given design process objectives. Such information is modelled in GDM by the information type design process evaluations. Two types are distinguished: epistemic design process performance information, and design process objective evaluations.

Epistemic (i.e., meta-level truth) information about performance indicators of the design process is modelled by the information type epistemic design process performance information, and evaluations of the given design process objectives by the information type design process objective evaluations. (Note that the use of epistemic information makes it is possible to state explicitly that it is *not* known what the value of a specific performance indicator of the design process is.) See Table 6.4 for the design process evaluations in Example 6.2.

TABLE 6.4. Design process evaluations in Example 6.2.

Kind	*Content*
Epistemic design process performance information	It is known that the design task has been completed in 153 hours.
Design process objective evaluation	The design process objective DPO1 is satisfied.

RQS. A characteristic feature of non-routine design processes is that they produce new requirement qualification sets. Such design processes start with relatively abstract, incomplete, ambiguous or contradictory design requirements that must be modified, completed and/or made more concrete before a fully satisfactory design object description can be generated. See Table 6.5 for the requirement qualification set (named RQS2) in Example 6.2.

TABLE 6.5. Requirement qualification set in Example 6.2.

Kind	*Content*
Requirement	The bicycle's price must be less than 400 euro. [R4]
Requirement	The bicycle must have light reflectors. [R5]
Qualified requirement	It is necessary to satisfy R4. [QR3]
Qualified requirement	It is necessary to satisfy R5. [QR4]
RQS	RQS2 includes design requirement information about the requirements R4 and R5 as well as the qualified requirements QR3 and QR4.

RQS assessments. A design process produces information about the satisfaction of design requirements and the fulfilment of requirement qualification sets. These *requirement qualification set assessments* are modelled in GDM by the information type RQS assessments. Two types are distinguished: design requirement assessments, and overall RQS assessments.

Design requirement assessments provide information about whether or not it is possible to satisfy specific sets of design requirements. This information is modelled by the information type design requirement assessments. *Overall RQS assessments* provide information about

whether or not it is possible to fulfil specific requirement qualification sets. This information is modelled by the information type overall RQS assessments. See Table 6.6 for the requirement qualification set assessments in Example 6.2.

TABLE 6.6. Requirement qualification set assessments in Example 6.2.

Kind	*Content*
Design requirement assessment	The requirement R4 can be satisfied.
Design requirement assessment	The qualified requirement QR3 can be satisfied.
Overall RQS assessment	The new set of design requirements RQS2 can be fulfilled.

DOD. Earlier in this section, a *design object description* has been described as a possible part of the input of a design process. A characteristic feature of design processes is that they intend to produce a design object description as output. See Table 6.7 for the design object description (named DOD2) in Example 6.2.

TABLE 6.7. Design object description in Example 6.2.

Kind	*Content*
DOD	DOD2 includes the domain object information that (a) the bicycle costs 350 euro, (b) the bicycle has light reflectors and (c) the wheels are made of stainless steel.

DOD assessments. A design process produces information about design object descriptions satisfying design requirements and fulfilling requirement qualification sets. These *design object description assessments* are modelled in GDM by the information type DOD assessments. Two types are distinguished: basic DOD assessments, and overall DOD assessments.

Basic DOD assessments denote information about the satisfaction of specific design requirements by specific design object descriptions. This information is modelled by the information type basic DOD assessments. *Overall DOD assessments* provide information about the fulfilment (or failure to fulfil) specific requirement qualification sets by specific design object descriptions. This information is modelled by the information type overall DOD assessments. See Table 6.8 for the design object description assessments in Example 6.2.

TABLE 6.8. Design object description assessments in Example 6.2.

Kind	*Content*
Basic DOD assessment	DOD2 satisfies requirement R4.
Overall DOD assessment	DOD2 fulfils RQS2.

Design process co-ordination

A design process co-ordination process controls a design process in accordance with given design process objectives. It imposes an overall design strategy on the requirement qualification set manipulation process and the design object description manipulation process.

In different applications, design process co-ordination may have different roles. For example, in a configuration process as described in Chapter 10, part of the design requirements is formed by customer specifications. Design process co-ordination orders to interrupt the design object description manipulation process only when the customer's specifications must be relaxed in order to be able to generate a satisfactory configuration.

In some other applications, design process co-ordination has a more prominent role. For example, in an aircraft re-design process as described in Chapter 13, whether or not to continue the design process and if so, what to do next, is often subject of deliberation. Design process co-ordination reconsiders the overall design strategy each time the requirement qualification set manipulation process or the design object description manipulation process has terminated its activation.

Figure 6.2 shows the types of information that a design process co-ordination process uses as input and the types of information that it produces as output.

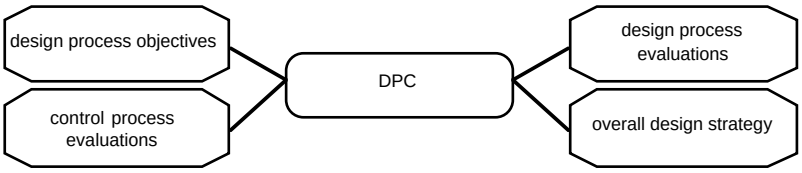


FIGURE 6.2. Input and output of a design process co-ordination process.

The following example, extending Example 6.1, is used to explain the types of input information used by a design process co-ordination process.

Example 6.3.
"The design task must be completed within 160 hours. The overall design strategy to use earlier design cases has been successful for RQS manipulation and has failed for DOD manipulation."

Design process objectives. Earlier in this section, design process objectives have been described as part of the input of a design process. Design process co-ordination operates on the design process objectives that are input for the design process.

Control process evaluations. *Control process evaluations* provide information about advances made by a design (sub-)process with respect to a specific overall design strategy. This information is modelled in GDM by the information type control process evaluations. Two types of control process evaluations are distinguished: evaluations of a requirement qualification set manipulation process (RQSM process evaluations), and evaluations of a design object description manipulation process (DODM process evaluations).

RQSM process evaluations provide evaluations of a requirement qualification set manipulation process, in terms of whether the process has succeeded or failed to act according to a specific overall design strategy. This information is modelled by the information type RQSM process evaluations. *DODM process evaluations* provide evaluations of a design object description manipulation process, in terms of whether the process has succeeded or failed to act according to a specific overall design strategy. This information is modelled by the information type DODM process evaluations. See Table 6.9 for the control process evaluations in Example 6.3.

TABLE 6.9. Control process evaluations in Example 6.3.

Kind	*Content*
RQSM process evaluation	The overall design strategy ODS1 was successful for RQS manipulation.
DODM process evaluation	The overall design strategy ODS1 has failed for DOD manipulation.

The following example, extending Example 6.3, is used to explain the types of output information produced by a design process co-ordination process.

Example 6.4.
"The design task is still in progress; so far, it has taken 40 hours, which is less than the given maximum of 160 hours, as required. The new overall design strategy is to leave the new set of design requirements as is and to generate a design object description from scratch; this may take at most 40 hours."

Design process evaluations. Earlier in this section, the design process evaluations produced by design process co-ordination have been described as part of the output of a design process. Design process co-ordination is responsible for the design process evaluations produced by a design process. See Table 6.10 for the design process evaluations in Example 6.4.

TABLE 6.10. Design process evaluations in Example 6.4.

Kind	*Content*
Epistemic design process performance information	So far, the design process has taken 40 hours.
Design process objective evaluation	Whether process objective PO1 is satisfied or violated is, as yet, undecided.

Overall design strategy. An *overall design strategy* establishes a strategy for the design process, which determines more detailed strategies for the manipulation of requirement qualification sets and design object descriptions, respectively. In GDM, the information type overall design strategy is used to model overall design strategies. (For the purpose of reference, each overall design strategy has a name that is unique in the context of a design process.) See Table 6.11 for the overall design strategy (named ODS2) in Example 6.4.

TABLE 6.11. Overall design strategy in Example 6.4.

Kind	*Content*
Overall design strategy	Leave RQS2 as is and generate a design object description from scratch; this may take at most 40 hours. [ODS2]

Requirement qualification set manipulation

On the basis of a given requirement qualification set, and in interaction with stake-holders (such as a client), a requirement qualification set manipulation process aims to generate a well defined requirement qualification set that includes sufficient design requirement information for the generation of a satisfactory design object description. This process always operates on one (possibly partial) set of design requirements called the *current requirement qualification set*. During a requirement qualification set manipulation process, the contents of the current requirement qualification set may vary due to the addition, modification, or deletion of design requirement information.

In different applications, requirement qualification set manipulation has different roles. For example, in a configuration process as described in Chapter 10, the role of requirement qualification set manipulation is simple. Each requirement is expressed as a triplet, consisting of a parameter value, a comparison operator and a value (e.g., "The platform running clearance equals 1.25 inches."). If a requirement needs to be relaxed, requirement qualification set manipulation just copies the original requirement and changes the value.

In other applications, requirement qualification set manipulation has a more complex role. For example, in an aircraft re-design process as described in Chapter 13, requirement qualification set manipulation applies different methods to generate and modify requirement qualification sets. One of these methods is to search a historical record of earlier design processes and retrieve design requirements generated in the context of an earlier, similar design task. If one of these old design requirements appears to be inconsistent with a design requirement included in the current requirement qualification set, then requirement qualification set manipulation resolves the conflict by deleting the old design requirement.

Figure 6.3 shows the types of information that a requirement qualification set manipulation process uses as input and the types of information that it produces as output. These types have been introduced earlier in this section.

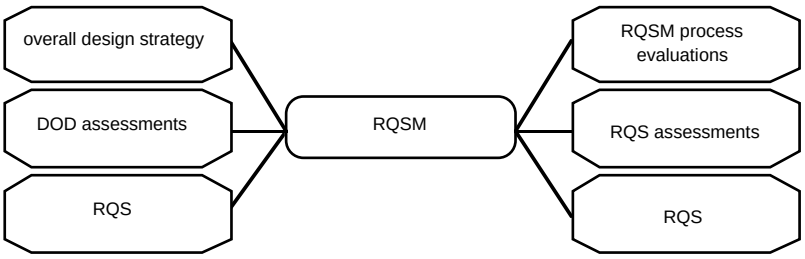


FIGURE 6.3. Input and output of a requirement qualification set manipulation process.

Design object description manipulation
A design object description manipulation process aims to generate a consistent design object description that fulfils a given requirement qualification set and that includes sufficient domain object information for the intended use of the design object description. (The intended use of a design object description is to be the basis for the assembly, construction, fabrication or another form of implementation of the design object.) This process always operates on one (possibly partial) description, called the *current design object description.* During a design object description manipulation process, the contents of the current design object description may vary due to the addition, modification, or deletion of domain object information.

There are many methods that design object description manipulation may apply during a design process. For example, in a configuration process as described in Chapter 10, the task of design object description manipulation is to make a satisfactory configuration of an elevator by selecting appropriate components and generating appropriate values for the parameters of these components. In an aircraft re-design process as described in Chapter 13, one method applied by design object description manipulation is to use an existing design object description generated for a similar type of aeroplane that has been developed earlier and (most importantly) that has been officially certified.

Figure 6.4 shows the types of information that a design object description manipulation process uses as input and the types of information that it produces as output. These types have been introduced earlier in this section.

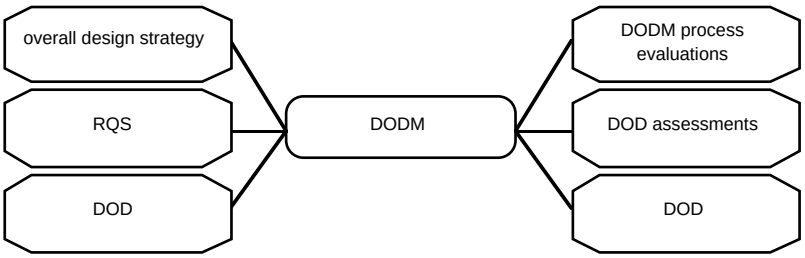


FIGURE 6.4. Input and output of a design object description manipulation process.

6.1.1.2 Process abstraction levels

A design process has been described to be composed of three processes: a design process coordination process to generate a successful overall design strategy, a requirement qualification set manipulation process to generate a suitable set of design requirements, and a design object description manipulation process to generate a satisfactory design object description. In GDM, three abstraction relations are used to model this composition; Figure 6.5 shows these relations between the component design and the respective components DPC, RQSM and DODM.

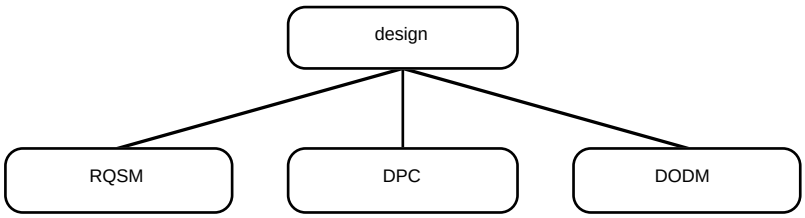


FIGURE 6.5. Two highest levels of abstraction for a design process.

6.1.2 Composition of processes

This section describes the way in which the higher-level processes involved in design are composed of lower-level processes, in terms of possibilities for exchange of information between processes, and in terms of task control knowledge used to control both the processes and the information exchange.

6.1.2.1 Information exchange

All four processes involved in a design process (i.e., design, design process co-ordination, requirement qualification set manipulation, and design object description manipulation) exchange information. As in Chapter 5, a distinction is made between information exchange between processes at different levels of process abstraction and information exchange between processes at the same level.

Firstly, a design process transfers the information it receives as input to its sub-processes (see Table 6.12). It transfers given design process objectives to DPC, given requirement qualification sets to RQSM, and given design object descriptions to DODM.

TABLE 6.12. Information transfer from a design process to its sub-processes.

Source	*Destination*	*Information link*	*Information type*
Design	DPC	Given design process objectives	Design process objectives
Design	RQSM	Given RQS	RQS
Design	DODM	Given DOD	DOD

Secondly, the three sub-processes of design exchange information with each other (see Table 6.13). Overall design strategies are transferred from DPC to RQSM and DODM. Control process evaluations are transferred from RQSM and DODM to DPC. Intermediate requirement qualification sets are transferred from RQSM to DODM, and intermediate DOD assessments from DODM to RQSM.

TABLE 6.13. Information exchange between sub-processes of a design process.

Source	*Destination*	*Information link*	*Information type*
DPC	RQSM	Intermediate overall design strategy to RQSM	Overall design strategy
DPC	DODM	Intermediate overall design strategy to DODM	Overall design strategy
RQSM	DPC	Intermediate RQSM process evaluations	RQSM process evaluations
RQSM	DODM	Intermediate RQS	RQS
DODM	DPC	Intermediate DODM process evaluations	DODM process evaluations
DODM	RQSM	Intermediate DOD assessments	DOD assessments

Thirdly, part of the information that the sub-processes produce as output is transferred to become output of the design process (see Table 6.14). Design process evaluations are transferred from DPC, requirement qualification sets and their assessments from RQSM, and design object descriptions and their assessments from DODM.

TABLE 6.14. Information transfer to a design process from its sub-processes.

Source	*Destination*	*Information link*	*Information type*
DPC	Design	Resulting design process evaluations	Design process evaluations
RQSM	Design	Resulting RQS assessments	RQS assessments
RQSM	Design	Resulting RQS	RQS
DODM	Design	Resulting DOD assessments	DOD assessments
DODM	Design	Resulting DOD	DOD

The possibilities for information exchange between processes involved in design, as shown in Tables 6.12 to 6.14, are modelled in GDM by information links. Figure 6.6 shows a pictorial representation of the information links within the component design.

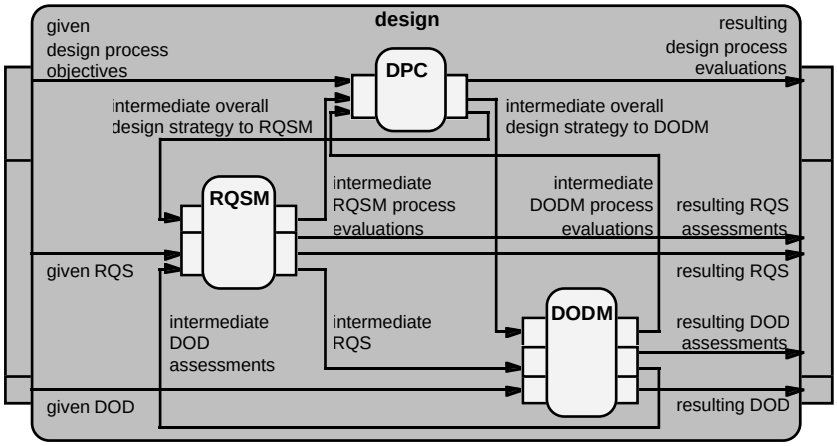


FIGURE 6.6. Information exchange between processes involved in design.

6.1.2.2 Task control

There are different ways in which a design process can be controlled. The generic control method described in this section can be used effectively in any design application, but it is not necessarily the most efficient: in specific situations, other methods may be more suitable.

Table 6.15 presents a summary of the method's implementation of the task control for a design process, given its sub-processes and possibilities for information exchange.

TABLE 6.15. Task control for a design process.

State	*Information link(s) updated next*	*Component(s) activated next*
Design has started.	Given design process objectives, given RQS, and given DOD.	DPC.
DPC has terminated and has succeeded to determine an overall design strategy.	Intermediate overall design strategy to RQSM, and intermediate overall design strategy to DODM.	RQSM, DODM.
RQSM and DODM have terminated.	Intermediate RQSM process evaluations, intermediate DODM process evaluations, intermediate RQS, and intermediate DOD assessments.	DPC.
DPC has terminated and has failed to determine an overall design strategy.	Resulting design process evaluations, resulting RQS, resulting RQS assessments, resulting DOD assessments, and resulting DOD.	None (design terminates).

Start of a design process

If a design process starts for the first time or when it is re-activated, receiving new information as input, then the following actions are taken:

- given design process objectives are transferred from the input of the design process to the input of design process co-ordination,
- given requirement qualification sets are transferred from the input of the design process to the input of RQS manipulation,
- given design object descriptions are transferred from the input of the design process to the input of DOD manipulation,
- design process co-ordination is activated to determine an overall design strategy.

This control knowledge is modelled in GDM by task control knowledge within the component design, declaring that if the component design is in its starting state, then in its next state component DPC will have been activated and the mediating links given design process objectives, given RQS and given DOD will have been updated. An initial target within the component DPC models the aim to produce an overall design strategy.

Termination of design process co-ordination
When design process co-ordination has terminated its activity, which actions are taken next depends in part on whether design process co-ordination has produced an overall design strategy or not. If design process co-ordination has succeeded to produce an overall design strategy, then the following actions are taken:

- the current overall design strategy is transferred from the output of design process co-ordination to the input of RQS manipulation and DOD manipulation, respectively,
- RQS manipulation and DOD manipulation are both activated to generate new requirement qualification sets and design object descriptions, respectively, and to report control process evaluations about their efforts to complete the current overall design strategy.

This control knowledge is modelled in GDM by task control knowledge within the component design, declaring that if, in the current state of the component design, the component DPC is idle after having been active and has succeeded with respect to its initial target, then in the next state of design the components RQSM and DODM will have been activated and the private links intermediate overall design strategy to RQSM and intermediate overall design strategy to DODM will have been updated. Initial targets within the components RQSM and DODM model the intent of RQS manipulation and DOD manipulation to produce control process evaluations with respect to the current overall design strategy.

If design process co-ordination has failed to produce an overall design strategy, there is no reason to continue the design process. A well defined requirement qualification set may have been generated as well as a consistent design object description fulfilling this requirement qualification set, achieving the given design process objectives. Alternatively, all attempts may have failed so far and design process co-ordination expects any further attempt (if at all possible) to fail as well. In all cases, the following actions are taken:

- the final design process evaluations are transferred from the output of design process co-ordination to the output of the design process,
- the final requirement qualification sets and requirement qualification set assessments are transferred from the output of RQS manipulation to the output of the design process,
- the final design object descriptions and design object description assessments are transferred from the output of DOD manipulation to the output of the design process,
- the design process is stopped.

This control knowledge is modelled in GDM by task control knowledge within the component design, declaring that if, in the current state of the component design, the component DPC is idle after having been active and has failed with respect to its target, then in the next state of design the mediating links resulting design process evaluations, resulting RQS assess-

ments, resulting RQS, resulting DOD assessments and resulting DOD will have been updated and the component design will have stopped.

Termination of RQS manipulation and DOD manipulation
When both RQS manipulation and DOD manipulation have produced control process evaluations, the following actions are taken:

- the available control process evaluations are transferred from the output of RQS manipulation and DOD manipulation, respectively, to the input of design process co-ordination,
- the available requirement qualification set is transferred from the output of RQS manipulation to the input of DOD manipulation,
- the available design object description assessments are transferred from the output of DOD manipulation to the input of RQS manipulation,
- design process co-ordination is activated to re-establish an overall design strategy.

This control knowledge is modelled in GDM by task control knowledge within the component design, declaring that if, in the current state of the component design, the components RQSM and DODM are both idle after having been active, then in the next state of design the component DPC will have been activated and the private links intermediate RQSM process evaluations, intermediate DODM process evaluations, intermediate RQS and intermediate DOD assessments will have been updated. The same initial target of DPC, as explained before, applies to model the aim of design process co-ordination to produce an overall design strategy.

6.2 Knowledge Composition

This section describes knowledge structures identified in design processes at different levels of abstraction and the composition of these knowledge structures. First the levels of reflection are described, and then the structure and composition of the knowledge identified at each of these levels.

6.2.1 Reflection levels

In a design process, the following four reflection levels are distinguished:

- The *object level* includes information about the behaviour, function, form and structure of domain objects. This level also includes domain-specific knowledge about the design object domain, such as (physical) laws.

- The *first meta-level*, directly above the object level, includes information about design requirements, design object descriptions, and their relations. This level also includes deductive knowledge to derive implicit design requirements, and knowledge to assess design object descriptions with respect to specific design requirements.

- The *second meta-level*, directly above the first meta-level, includes information about requirement qualification sets and about the relations between requirement qualification sets and design object descriptions. This level also includes knowledge to assess design object descriptions with respect to specific requirement qualification sets, and strategic knowledge to determine modifications of requirement qualification sets and design object descriptions, respectively.

- The *third meta-level*, directly above the second meta-level, includes information about design process objectives and about design process evaluations. This level also includes deductive knowledge to derive implicit design process objectives, and strategic knowledge to determine overall design strategies and strategies for the manipulation of requirement qualification sets and design object descriptions, respectively.

6.2.2 Knowledge structures and composition

This section describes the composition and structure of the generic knowledge related to design and shows how they are modelled in GDM. For the sake of comprehension, in the figures hereafter the boxes with thick lines indicate those parts of GDM that are candidates for refinement. (When using GDM to model design processes in a specific application domain, some of the knowledge structures need to be specialised or instantiated.)

The presented examples extend the bicycle design example. In these examples, terms and expressions shown in italics are part of the model of the application domain, not of GDM.

6.2.2.1 Structure and composition of knowledge at the object level

The object level includes *domain object information* about the behaviour, function, form and structure of domain objects in the universe of discourse. Figure 6.7 shows the composition of the information type domain object information, which models such information about domain objects. In the following, the information types are explained to which the information type domain object information refers.

The information type domain object type models domain objects in the universe of discourse. Two types of domain objects are distinguished: domain object constants, and domain object variables.

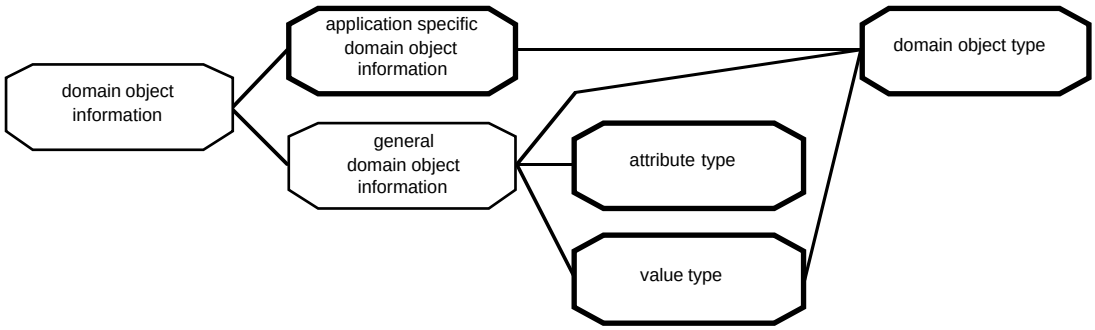


FIGURE 6.7. Composition of information type domain object information.

Domain object constants refer to specific domain objects. They are modelled by objects of the sort domain-object-constant, which is a sub-sort of the sort domain-object. (That is, every object of the sort domain-object-constant may be used in every function and each relation with an argument of the sort domain-object.) In practice, this concept is useful to model an explicit reference to a specific domain object. In the bicycle example, the objects *bicycle1*, *frame1*, and *motion-transfer-system1* from the sort domain-object-constant specify a specific bicycle, frame and motion transfer system, respectively.

Domain object variables refer to non-specific domain objects. They are modelled by objects of the sort domain-object-variable, which is a sub-sort of the sort domain-object. This concept is useful to model an implicit reference to an unnamed domain object; such a reference is used in the definitions of requirements (to be explained later).

The information type attribute type models attributes (or, properties) of domain objects, such as colour, length and weight. In principle, any domain object attribute can be specified by means of an object of the sort attribute.

The information type value type models values of the attributes of domain objects, such as blue (for colour), 1.86 meter (for length) and 80 kg (for weight). In principle, any distinct value can be specified by means of an object of the sort value.

Figure 6.8 shows the structure of the information type general domain object information, which models information about the values of specific attributes of specific domain objects. This information type contains two relations: the relation has-value has three arguments of the sorts domain-object, attribute, and value, respectively; and the relation has-part has two arguments that are both of the sort domain-object.

An atom has-value(*domain-object1*, *attribute1*, *value1*) specifies that the attribute *attribute1* of domain object *domain-object1* has value *value1*. This relation can be used to express any discrete behavioural pattern, function, or form of domain objects. An atom has-part(*domain-object1*, *domain-object2*) specifies that domain object *domain-object2* is a part of domain object *domain-object1*. This relation can be used to express the structure of complex domain objects.

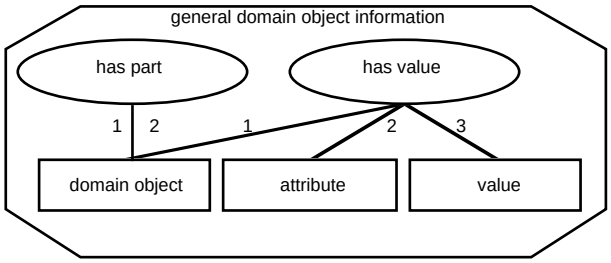


FIGURE 6.8. Structure of information type general domain object information.

Example 6.5.
"The bicycle has a six-gear motion-transfer system."

has-part(*bicycle1*, *motion-transfer-system1*)
has-value(*motion-transfer-system1*, *number-of-gears*, 6)

The information type application specific domain object information models application specific information about domain objects (i.e., information that is specific for the domain of application and/or the type of design process). Note that such information can also be modelled by means of the information type general domain object information, but in practice it may sometimes be more convenient to use self-defined relations. For instance, in Example 6.5 the fact that the motion transfer system has six gears could also have been modelled by means of the application specific atom *has-number-of-gears*(*motion-transfer-system1*, 6).

6.2.2.2 Structure and composition of knowledge at the first meta-level

The first meta-level includes the following main information types: (1) design object descriptions, (2) information about the design requirements of a design object, (3) basic assessments of specific design object descriptions, and (4) assessments of specific design requirements.

Design object descriptions. Figure 6.9 shows the composition of the information type DOD, which models epistemic information about the contents of specific design object descriptions. That is, this information type may be used to model the partial domain object information included in a design object description.

The information type DOD name type models names of design object descriptions. Each design object description groups information that, as a whole, describes in full or in part the behaviour, function, form, or structure of a design artefact and other relevant domain objects. A design object description is uniquely identified by its name within a design process.

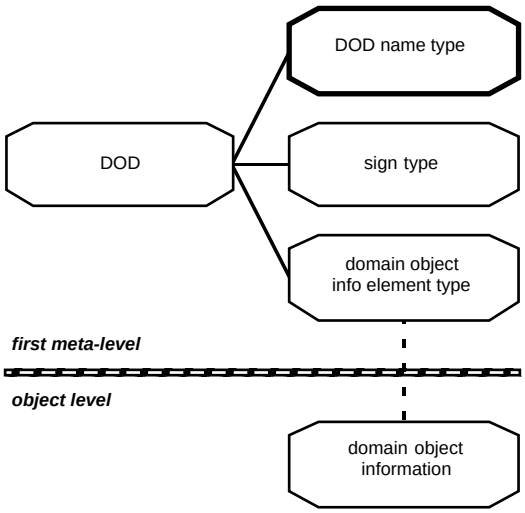


FIGURE 6.9. Composition of information type DOD.

The information type domain object info element type models meta-descriptions of domain object information. These meta-descriptions are obtained by the upward reflection of domain object information (modelled by the information type domain object information) and are used to formulate epistemic information about domain object information, goals for the deduction of new domain object information, assumptions about domain object information, and requirements of a design artefact.

The information type sign type models partial truth values. The sort sign consists of three objects: pos specifies the truth value *true*, neg specifies *false*, and unk specifies *unknown*.

Figure 6.10 shows the structure of the information type DOD. This information type contains the relation includes-domain-object-information, which has three arguments of the sorts DOD-name, domain-object-info-element, and sign, respectively. An atom includes-domain-object-information(*DOD-name1*, *domain-object-info-element1*, *sign1*) specifies the information that the design object description named *DOD-name1* includes the domain object information *domain-object-info-element1*, with sign *sign1* as its truth value.

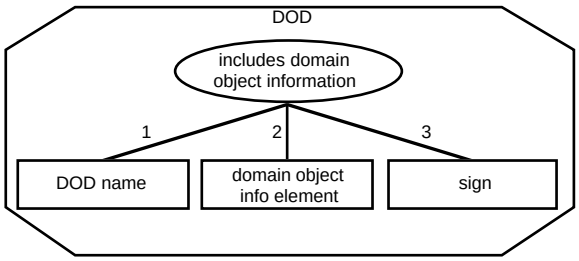


FIGURE 6.10. Structure of information type DOD.

Example 6.6.
"The design object description named DOD1 includes the information that the bicycle's frame contains shock absorbers and is not safe to be used by young children, and that it is unknown whether the bicycle is suited for riding in mountains or not."

includes-domain-object-information(*DOD1*, has-part(*bicycle1*, *frame1*), pos)
includes-domain-object-information(*DOD1*, *contains*(*frame1*, *shock-absorbers*), pos)
includes-domain-object-information(
DOD1, *is-safe-to-be-used-by*(*bicycle1*, *young-children*), neg)
includes-domain-object-information(
DOD1, *is-suitable-for-terrain-type*(*bicycle1*, *mountains*), unk)

Design requirement information. Figure 6.11 shows the composition of the information type design requirement information, which models information about design requirements.

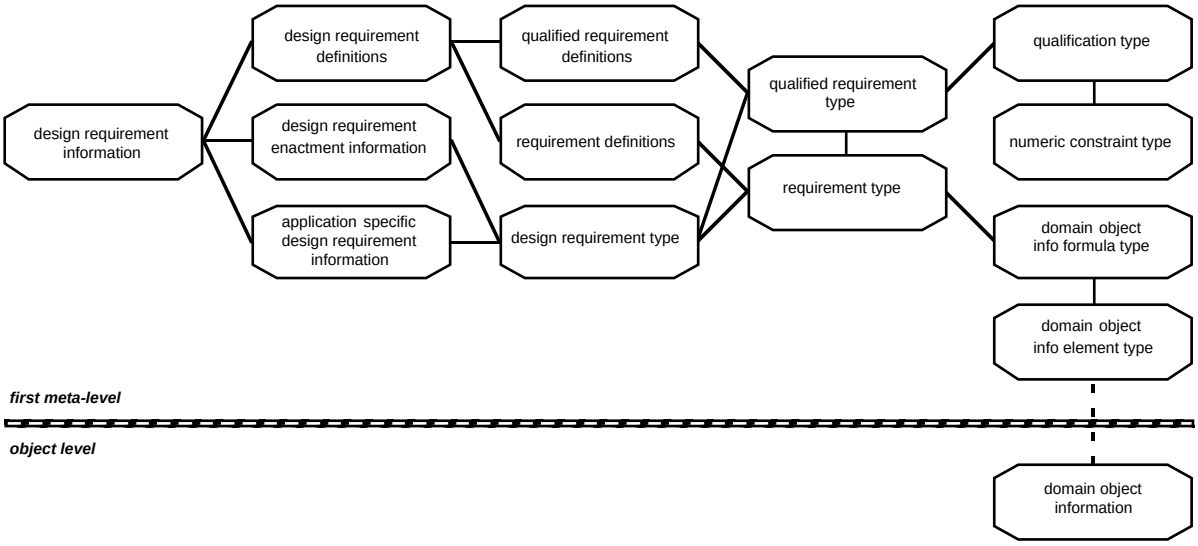


FIGURE 6.11. Composition of information type design requirement information.

The information type design requirement information refers to three information types: (1) design requirement definitions, which models definitions of design requirements, (2) design requirement enactment information, which models information about which design requirements must actually be satisfied in a given situation, and (3) application specific design requirement information, which models application specific information about design requirements.

The information type design requirement definitions refers to the information types requirement definitions and qualified requirement definitions, which model definitions of requirements and definitions of qualified requirements, respectively. Together, requirements and qualified

requirements are termed *design requirements*, which are modelled by the information type design requirement type.

A *requirement* specifies what the behaviour, function, form, or structure of the design object should be. Objects of the sort requirement within the information type requirement type are used to model requirements. A requirement may be formulated directly in terms of a domain object information formula (modelled by an object of the sub-sort domain-object-info-formula) or indirectly by means of a requirement name (modelled by an object of the sub-sort requirement-name).

A *domain object information formula* is expressed in the domain object language, which is used to describe the properties of the design artefact and other domain objects and the relations between domain objects. A domain object information formula is composed of a finite number of domain object information elements (modelled by objects of the sort domain-object-info-element) and the following functions for different predicate-logical operators:

- l-not for negation ($\neg$),
- l-or for disjunction ($\vee$),
- l-and for conjunction ($\wedge$),
- l-implies for implication ($\Rightarrow$),
- exists for existential quantification ($\exists$), and
- for-all for universal quantification ($\forall$).

A *requirement name* is used to refer to a requirement; it is application specific and assumed to be unique within the context of a design process. Information about the domain object information formula corresponding to a given requirement name is modelled by the information type requirement definitions.

Figure 6.12 shows the structure of the information type requirement definitions. It contains the relation is-defined-as, which has two arguments of the sorts requirement-name and domain-object-info-formula, respectively. An atom is-defined-as(*requirement-name1*, *domain-object-info-formula1*) specifies that the requirement named *requirement-name1* is defined as the domain object information formula *domain-object-info-formula1* on the behaviour, function, form or structure of the design artefact.

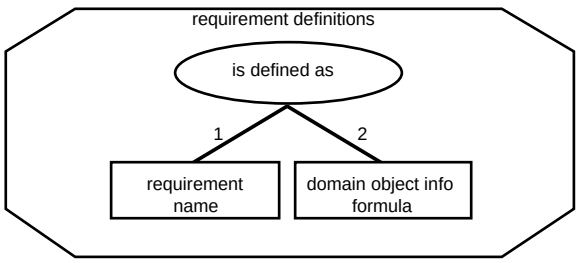


FIGURE 6.12. Structure of information type requirement definitions.

Example 6.7.
"The bicycle must be suited for riding in mountains (requirement R1), low-priced (requirement R2) and safe to be used by young children (requirement R3)."

is-defined-as(*R1*, *is-suitable-for-terrain-type*(*bicycle1*, *mountains*))
is-defined-as(*R2*, *is-price-level*(*bicycle1*, *low*))
is-defined-as(*R3*, *is-safe-to-be-used-by*(*bicycle1*, *young-children*))

A *qualified requirement* specifies the (relative or absolute) importance of satisfying a list of given requirements. Objects of the sort qualified-requirement within the information type qualified requirement type are used to model qualified requirements. A qualified requirement may be formulated directly as a qualified requirement expression (modelled by an object of the sub-sort qualified-requirement-expression) or indirectly by means of a qualified requirement name (modelled by an object of the sub-sort qualified-requirement-name).

A *qualified requirement expression* is composed of a qualification (modelled by an object of the sort qualification) and a requirement list (modelled by an object of the sort requirement-list). This composition is specified by the function expr, which maps a pair of objects from the sort qualification and the sort requirement-list on an object of the sort qualified-requirement-expression.

A *qualification* is used to express a constraint on the number of requirements to be satisfied from the list, as well as information about whether the order of the requirements in the list is relevant for satisfaction or not. The sort qualification includes an object every; when used in a qualified requirement expression with a list *L* of requirements, the object every specifies that every requirement from *L* must be satisfied.

The sort qualification also contains two functions, at-random and in-preferred-order, which both have one argument of the sort numeric-constraint. The information type numeric constraint type in which this sort is defined, is used to model *numeric constraints*. It contains the functions at-least, exactly, and at-most, each mapping objects from the sort integer to objects from the sort numeric-constraint, and with an intuitive meaning.

When used in a qualified requirement expression with a list *L* of requirements, the term at-random(*numeric-constraint1*) specifies that the number of requirements from *L* to be satisfied is constrained by *numeric-constraint1*, as well as that the order of the requirements in *L* has no significance for satisfaction. The term in-preferred-order(*numeric-constraint1*) specifies that the number of requirements from *L* to be satisfied is constrained by *numeric-constraint1*, as well as that the order of the requirements in *L* signifies which ones are more preferred to be satisfied.

A *qualified requirement name* is used to refer to a qualified requirement; it is application specific and assumed to be unique within the context of a design process. Information about the qualified requirement expression corresponding to a given qualified requirement name is modelled by the information type qualified requirement definitions.

Figure 6.13 shows the structure of the information type qualified requirement definitions. It contains the relation is-defined-as, which has two arguments of the sorts qualified-requirement-

name and qualified-requirement-expression, respectively. (Note that is-defined has been defined earlier with different arguments.) An atom is-defined-as(*qualified-requirement-name1*,*qualified-requirement-expression1*) specifies that the qualified requirement named *qualified-requirement-name1* is defined as the qualified requirement expression *qualified-requirement-expression1*.

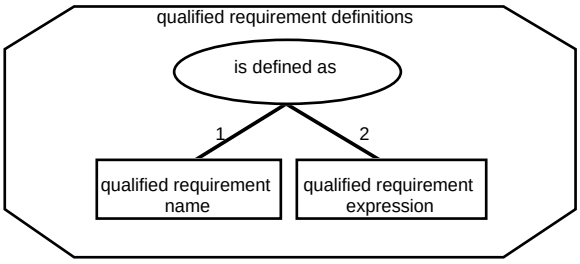


FIGURE 6.13. Structure of information type qualified requirement definitions.

Example 6.8.
"The qualified requirement named QR1 states that exactly three of the requirements R4, R5, R6, and R7 must be satisfied, but that there are no preferences between these requirements. The qualified requirement named QR2 states that at least one of the requirements R2 and R3 must be satisfied, and that satisfying requirement R3 is preferred to satisfying requirement R2."

is-defined-as(*QR1*, expr(at-random(exactly(3)), [*R4*, *R5*, *R6*, *R7*]))
is-defined-as(*QR2*, expr(in-preferred-order(at-least(1)), [*R3*, *R2*]))

Figure 6.14 shows the structure of the information type design requirement enactment information, which models information about which design requirements must actually be satisfied in a given situation. It contains the relation is-to-be-satisfied, which has one argument of the sort design-requirement. An atom is-to-be-satisfied(*design-requirement1*) specifies that design requirement *design-requirement1* is to be satisfied.

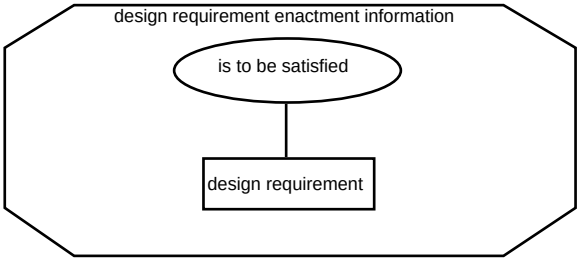


FIGURE 6.14. Composition of information type design requirement enactment information.

Example 6.9.
"The requirement R1 and the qualified requirements QR2 and QR3 must be satisfied."

is-to-be-satisfied(*R1*)
is-to-be-satisfied(*QR2*)
is-to-be-satisfied(*QR3*)

The information type application specific design requirement information models application specific information about design requirements. For example, in some types of design processes different design parties are involved, such as the customer, the designer, and the design support system. In such situations, it may be of interest to know which design party is the source of a specific design requirement. This information can be modelled by means of a relation *is-source-of* with two arguments of the sorts *design-party* and design-requirement, respectively, which is specified within the information type application specific design requirement information.

Basic DOD assessments. Figure 6.15 shows the composition of the information type basic DOD assessments, which models basic assessments of specific design object descriptions. Basic assessments denote content-based relations between design object descriptions as well as satisfaction relations between specific design object descriptions and (sets of) design requirements.

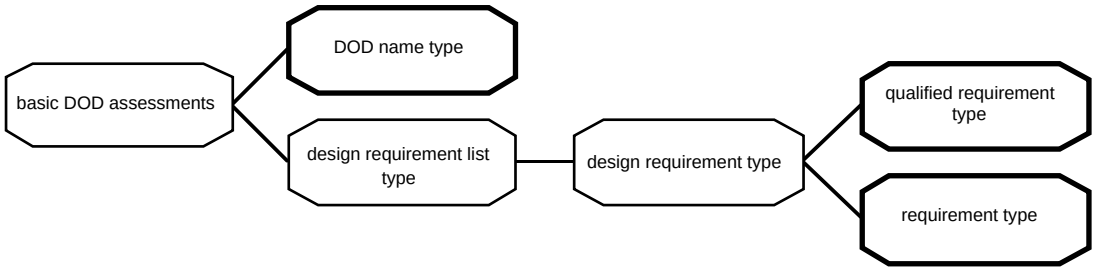


FIGURE 6.15. Composition of information type basic DOD assessments.

The information type design requirement list type specifies a sort design-requirement-list, which is used to model sets (or, unordered lists) of design requirements. The sort design-requirement is defined as a sub-sort of the sort design-requirement-list.

Figure 6.16 shows the structure of the information type basic DOD assessments. It contains six relations: the relations is-refinement-of and is-consistent-with each have two arguments that are both of the sort DOD-name, and the relations satisfies, violates, can-be-refined-to-satisfy, and is-decisive-wrt-satisfaction-of each have two arguments of the sorts DOD-name and design-requirement-list, respectively.

An atom is-refinement-of(*DOD-name1*, *DOD-name2*) specifies that the design object description named *DOD-name1* is a refinement of the design object description named *DOD-name2*. That is, every domain object information element with truth value *true* or *false* that *DOD-name2* includes is also included in *DOD-name1*.

An atom is-consistent-with(*DOD-name1*, *DOD-name2*) specifies that the design object description named *DOD-name1* is consistent with the design object description named *DOD-name2*. That is, *DOD-name1* does not include any domain object information element with truth value *true* (*false*) that is included with truth value *false* (*true*) in *DOD-name2*.

An atom satisfies(*DOD-name1*, *design-requirement-list1*) specifies that the design object description named *DOD-name1* satisfies every design requirement from the design requirement list *design-requirement-list1*. If the atom is used in negated form, then it specifies that *DOD-name1* violates at least one of the design requirements from *design-requirement-list1* or that *DOD-name1* is indecisive with respect to *design-requirementlist1*.

An atom violates(*DOD-name1*, *design-requirement-list1*) specifies that the design object description named *DOD-name1* violates at least one design requirement from *design-requirement-list1*. If the atom is used in negated form, then it specifies that *DOD-name1* satisfies every design requirement from *design-requirement-list1* or that *DOD-name1* is indecisive with respect to *design-requirementlist1*.

An atom can-be-refined-to-satisfy(*DOD-name1*, *design-requirement-list1*) specifies that there exists a design object description that is a refinement of the design object description named *DOD-name1* and that satisfies every design requirement from *design-requirement-list1*. If the atom is used in negated form, then it specifies that every design object description that is a refinement of *DOD-name1* either violates at least one of the design requirements from *design-requirement-list1* or is indecisive with respect to *design-requirementlist1*.

An atom is-decisive-wrt-satisfaction-of(*DOD-name1*, *design-requirement-list1*) specifies that the design object description named *DOD-name1* is decisive with respect to the satisfaction of every design requirement from *design-requirement-list1*. That is, it is known that *DOD-name1* or one of its refinements either satisfies every design requirement from *design-requirement-list1* or violates at least one of the design requirements from *design-requirement-list1*.

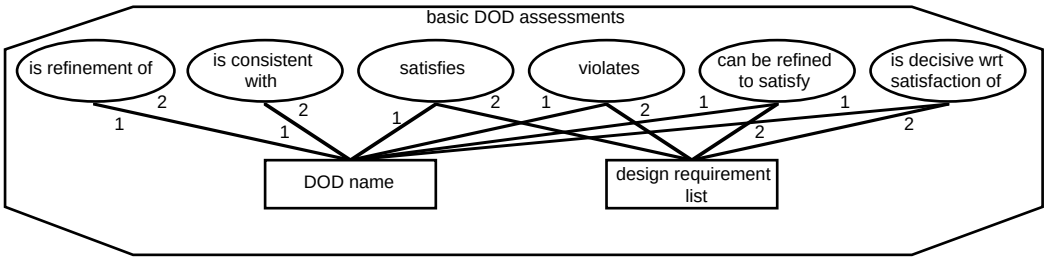


FIGURE 6.16. Structure of information type basic DOD assessments.

Example 6.10.
"The design object description named DOD3 is a refinement of the description named DOD2; they are also consistent with each other. The design object description named DOD1 satisfies qualified requirement QR2, it violates at least one of the requirements R2 and R3 and it is indecisive with respect to the satisfaction of requirement R1. However, there is a refinement of DOD2 that satisfies requirement R1.

is-refinement-of(*DOD3*, *DOD2*)
is-consistent-with(*DOD2*, *DOD3*)
satisfies(*DOD1*, *QR2*)
violates(*DOD1*, [*R2*, *R3*])
not is-decisive-wrt-satisfaction-of(*DOD1*, *R1*)
can-be-refined-to-satisfy(*DOD2*, *R1*)

Design requirement assessments. Figure 6.17 shows the composition of the information type design requirement assessments, which models assessments of design requirements. Such assessments denote satisfiability properties of design requirements, some of which may be derived from basic assessments of design object descriptions.

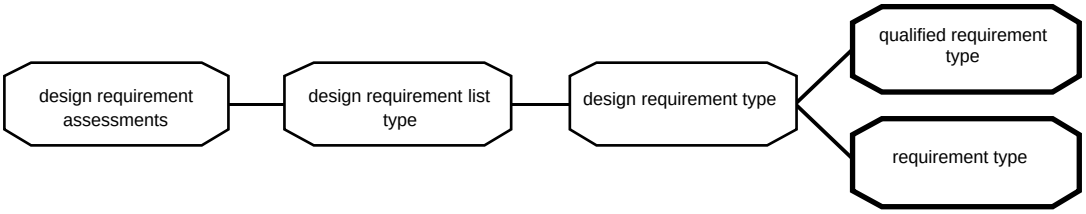


FIGURE 6.17. Composition of information type design requirement assessments.

Figure 6.18 shows the structure of the information type design requirement assessments. It contains four relations, is-satisfiable, is-tautological, is-contradictory, and is-imprecise, which each have one argument of the sort design-requirement-list. For each of these relations, the order of design requirements in the argument list is not relevant (i.e., the list can be treated as a set).

An atom is-satisfiable(*design-requirement-list1*) specifies that there is a design object description that satisfies the design requirements from the set *design-requirement-list1*. An atom is-tautological(*design-requirement-list1*) specifies that the set of design requirements *design-requirement-list1* is a tautology (i.e., is always true), so every design object description satisfies *design-requirement-list1*.

An atom is-contradictory(*design-requirement-list1*) specifies that the set of design requirements *design-requirement-list1* is a contradiction (i.e., is always false), that is, every design object description violates *design-requirement-list1*. An atom is-imprecise(*design-requirement-*

list1) specifies that there does not exist a design object description that, when restricted to the domain object information necessary for the construction of the design object, is decisive with respect to the satisfaction of the set of design requirements *design-requirement-list1*.

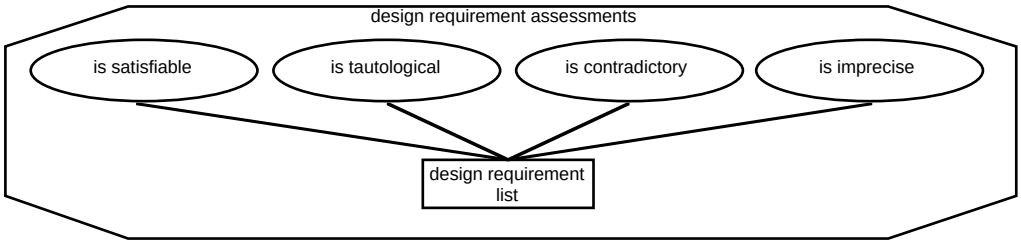


FIGURE 6.18. Structure of information type design requirement assessments.

Example 6.11.
"Requirement R2 can be satisfied. The requirement that the bicycle to be designed either has a one-piece frame or does not have a one-piece frame is trivially satisfied. Qualified requirement QR4 is contradictory, and the qualified requirement that the bicycle must be beautiful is imprecise."

is-satisfiable(*R2*)
is-tautological(l-or(*has-frame-type*(*bicycle1*, *1piece*), l-not(*has-frame-type*(*bicycle1*, *1piece*))))
is-contradictory(*QR4*)
is-imprecise(expr(every, [*has-appearance*(*bicycle1*, *beautiful*)]))

6.2.2.3 Structure and composition of knowledge at the second meta-level

The second meta-level includes the following main information types: (1) requirement qualification sets, (2) assessments of design object descriptions, (3) assessments of requirement qualification sets, and (4) information about the results of the design process.

Requirement qualification sets. Figure 6.19 shows the composition of the information type RQS, which models epistemic information about the contents of specific requirement qualification sets. That is, this information type may be used to model the partial design requirement information included in a requirement qualification set.

The information type RQS name type models names of requirement qualification sets. Each requirement qualification set groups information that, as a whole, describe in full or in part the design requirements of the design artefact and relevant relations between these design requirements. A requirement qualification set is uniquely identified by its name within a design process.

The information type design requirement info element type models meta-descriptions of design requirement information. These meta-descriptions are obtained by the upward reflection

of design requirement information (modelled by the information type design requirement information) and are used to formulate epistemic information about design requirement information, goals for the deduction of new design requirement information, and assumptions about design requirement information.

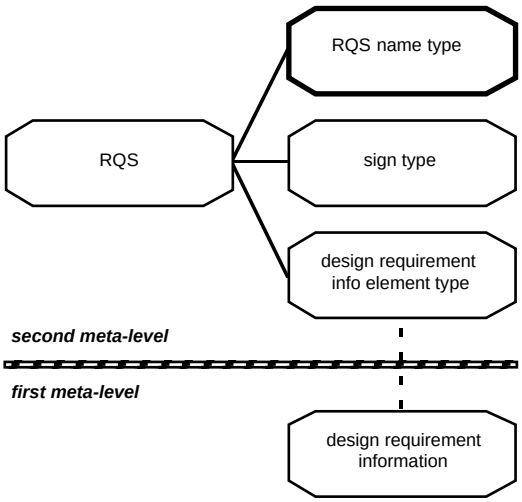


FIGURE 6.19. Composition of information type RQS.

Figure 6.20 shows the structure of the information type RQS. It contains the relation includes-design-requirement-information, which has three arguments of the sorts RQS-name, design-requirement-info-element, and sign, respectively. An atom includes-design-requirement-information(*RQS-name1*, *design-requirement-info-element1*, *sign1*) specifies the epistemic information that the requirement qualification set named *RQS-name1* includes the design requirement information *design-requirement-info-element1*, with sign *sign1* as its truth value.

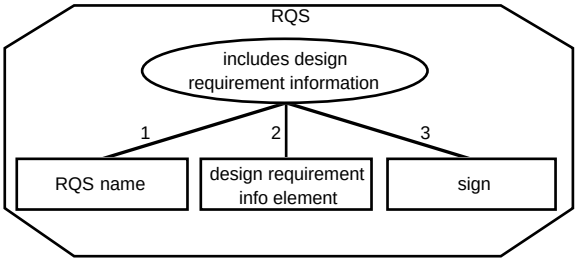


FIGURE 6.20. Structure of information type RQS.

Example 6.12.
"The requirement qualification set named RQS2 includes the information that the bicycle must be suited for riding in mountains (R1), that the bicycle must be safe to be used by young children (R3), that the bicycle's price must be less than 400 euro (R4), and that the bicycle must have light reflectors (R5). Furthermore, RQS2 includes the information that it is necessary to satisfy the requirements R1 and R5 (which is expressed by means of qualified requirement QR6) and that satisfying requirement R3 is preferred over satisfying requirement R4, if irreconcilable (which is expressed by means of qualified requirement QR7)."

includes-design-requirement-information(*RQS2*,
  is-defined-as(*R1*, *is-suitable-for-terrain-type*(*bicycle1*, *mountains*)), pos)
includes-design-requirement-information(*RQS2*,
  is-defined-as(*R3*, *is-safe-to-be-used-by*(*bicycle1*, *young-children*)), pos)
includes-design-requirement-information(*RQS2*,
  is-defined-as(*R4*, for-all(*P*, l-implies(has-value(*bicycle1*, *price*(*euro*), *P*), *P* < 400))), pos)
includes-design-requirement-information(*RQS2*,
  is-defined-as(*R5*, *has-as-accessory*(*bicycle1*, *light-reflectors*)), pos)
includes-design-requirement-information(*RQS2*, is-defined-as(*QR6*, every, [*R1*, *R5*]), pos)
includes-design-requirement-information(*RQS2*,
  is-defined-as(*QR7*, at-random(at-least(1)), [*R3*, *R4*]), pos)

DOD assessments. Figure 6.21 shows the composition of the information type DOD assessments, which models information about the assessment of design object descriptions. Two main types of DOD assessments are distinguished: (1) overall assessments (at the second meta-level) of specific design object descriptions regarding the fulfilment of specific requirement qualification sets, and (2) epistemic information about assessments (at the first meta-level) of specific design requirements and basic assessments of design object descriptions in relation to specific design requirements.

The information type basic evaluation info element type models meta-descriptions of design requirement assessments (modelled by the information type design requirement assessments) and basic DOD assessments (modelled by the information type basic DOD assessments). These meta-descriptions are obtained by upward reflection and are used to formulate epistemic information about design requirement assessments and basic DOD assessments.

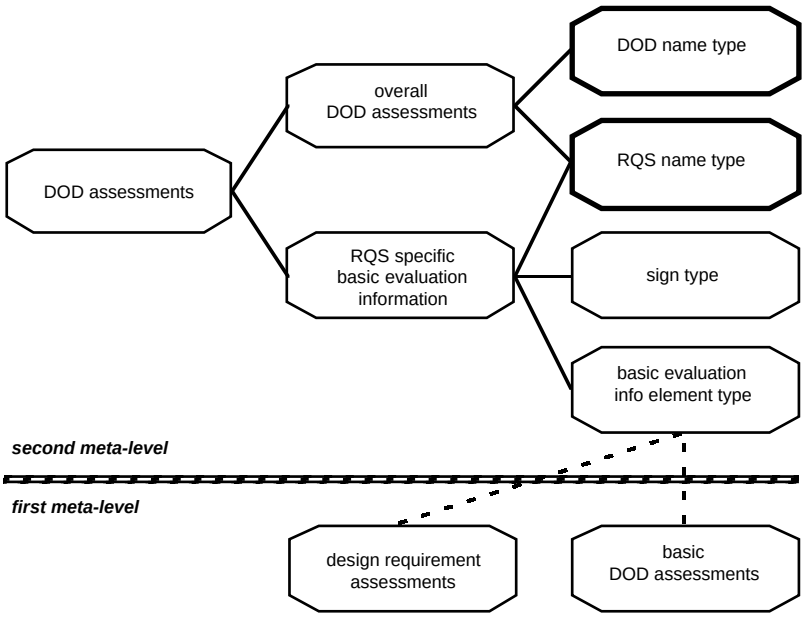


FIGURE 6.21. Composition of information type DOD assessments.

Figure 6.22 shows the structure of the information type RQS specific basic evaluation information, which models epistemic information related to specific requirement qualification sets about assessments of design requirements and basic assessments of design object descriptions. It contains the relation includes-basic-evaluation-information, which has three arguments of the sorts RQS-name, basic-evaluation-info-element, and sign, respectively. An atom includes-basic-evaluation-information(*RQS-name1*, *basic-evaluation-info-element1*, *sign1*) specifies that the requirement qualification set named *RQS-name1* includes the basic evaluation information *basic-evaluation-info-element1*, with sign *sign1* as its truth value.

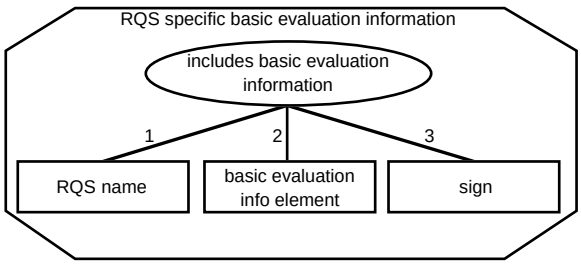


FIGURE 6.22. Structure of information type RQS specific basic evaluation information.

Example 6.12.
"The set of design requirements named RQS2 includes the basic evaluation information that the design object description named DOD2 satisfies the requirement that the bicycle's price should be less than 400 euro, that the design object description named DOD1 does not satisfy qualified requirement QR1, that requirement R2 can be satisfied, and that qualified requirement QR4 is contradictory."

includes-basic-evaluation-information(*RQS2*,
  satisfies(*DOD2*, for-all(*P*, l-implies(has-value(*bicycle1*, *price*(*euro*), *P*), *P* < 400)))), pos)
includes-basic-evaluation-information(*RQS2*, satisfies(*DOD1*, *QR1*), neg)
includes-basic-evaluation-information(*RQS2*, is-satisfiable(*R2*), pos)
includes-basic-evaluation-information(*RQS2*, is-contradictory(*QR4*), pos)

Figure 6.23 shows the structure of the information type overall DOD assessments, which models assessments of design object descriptions regarding the fulfilment of requirement qualification sets. It contains four relations, fulfils, fails-to-fulfil, can-be-refined-to-fulfil, and is-decisive-wrt-fulfilment-of, which each have two arguments of the sorts DOD-name and RQS-name, respectively.

An atom fulfils(*DOD-name1*, *RQS-name1*) specifies that the design object description named *DOD-name1* fulfils the requirement qualification set named *RQS-name1*. That is, *DOD-name1* satisfies each design requirement included in *RQS-name1* that is to be satisfied. (Note that this definition presumes the existence of design requirement enactment information as described in the previous sub-section.) An atom fails-to-fulfil(*DOD-name1*, *RQS-name1*) specifies that the design object description named *DOD-name1* fails to fulfil the requirement qualification set named *RQS-name1*. That is, *DOD-name1* violates at least one of the design requirements included in *RQS-name1* that is to be satisfied.

An atom can-be-refined-to-fulfil(*DOD-name1*, *RQS-name1*) specifies that there is a design object description that is a refinement of the design object description *DOD-name1* and that fulfils the requirement qualification set named *RQS-name1*. An atom is-decisive-wrt-fulfilment-of(*DOD-name1*, *RQS-name1*) specifies that the design object description named *DOD-name1* or one of its refinements either fulfils the requirement qualification set named *RQS-name1* or fails to fulfil *RQS-name1*.

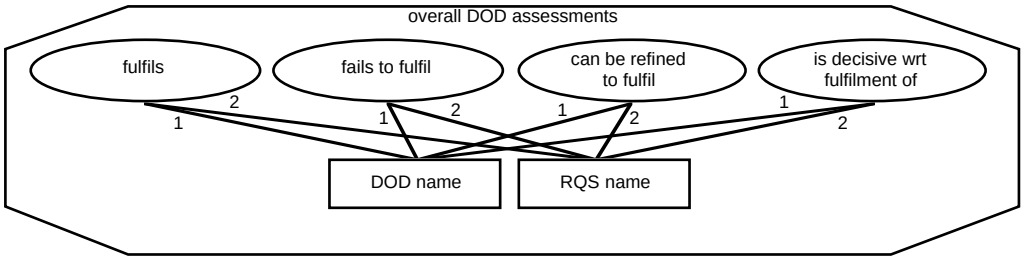


FIGURE 6.23. Structure of information type overall DOD assessments.

Example 6.13.
"Design object description DOD1 does not fulfil requirement qualification set RQS1, and design object description DOD2 fulfils requirement qualification set RQS2."

fails-to-fulfil(*DOD1*, *RQS1*)
fulfils(*DOD2*, *RQS2*)

RQS assessments. Figure 6.24 shows the composition of information type RQS assessments, which models information about the assessment of requirement qualification sets. Two main types of RQS assessments are distinguished: (1) overall assessments (at the second meta-level) of specific requirement qualification sets regarding their fulfilment, and (2) epistemic information about assessments (at the first meta-level) of specific design requirements and basic assessments of design object descriptions in relation to specific design requirements.

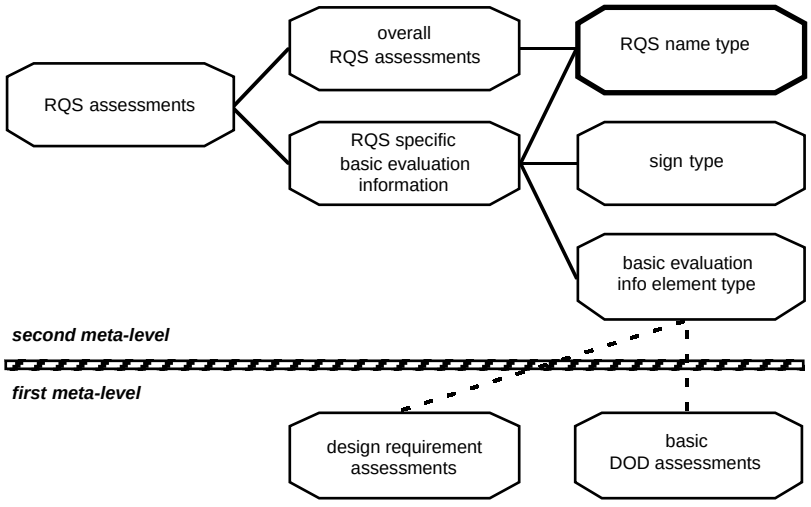


FIGURE 6.24. Composition of information type RQS assessments.

The information type basic evaluation info element type models meta-descriptions of design requirement assessments (modelled by the information type design requirement assessments) and basic DOD assessments (modelled by the information type basic DOD assessments). This information type has already been explained in relation to DOD assessments.

Figure 6.25 shows the structure of the information type overall RQS assessments, which models assessments of requirement qualification sets regarding their fulfilment. It contains five relations, can-be-fulfilled, is-inconsistent, is-ambiguous, is-imprecise, and is-incomplete, which each have one argument of the sort RQS-name. The definitions of the latter four relations are based on those from Smithers, Corne and Ross [Smithers, Corne and Ross, 1994].

An atom can-be-fulfilled(*RQS-name1*) specifies that there exists a design object description that fulfils the requirement qualification set named *RQS-name1*. An atom is-inconsistent(*RQS-name1*) specifies that the requirement qualification set named *RQS-name1* is inconsistent, meaning that any complete design object description fails to fulfil *RQS-name1*. An atom is-ambiguous(*RQS-name1*) specifies that the requirement qualification set named *RQS-name1* is ambiguous, meaning that there exist two or more complete design object descriptions that fulfil *RQS-name1*. An atom is-imprecise(*RQS-name1*) specifies that the requirement qualification set named *RQS-name1* is imprecise, meaning there does not exist a design object description that, when restricted to the included domain object information necessary for the construction of the design object, is decisive with respect to the fulfilment of *RQS-name1*. An atom is-incomplete(*RQS-name1*) specifies that the requirement qualification set named *RQS-name1* is incomplete, meaning that it is not equivalent to its deductive closure (and, therefore, needs to be further deductively refined).

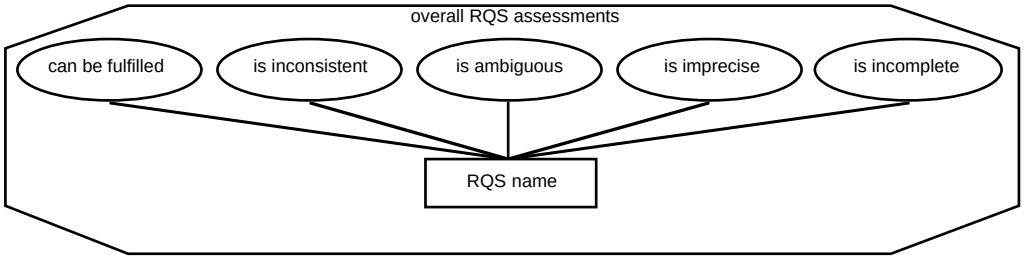


FIGURE 6.25. Structure of information type overall RQS assessments.

Example 6.14.
"The requirement qualification set named RQS2 can be fulfilled."

can-be-fulfilled(*RQS2*)

Design process results information. Figure 6.26 shows the composition of the information type design process results information, which models information about the (intermediate or final) results of a design process. Five main types are distinguished: (1) information about

requirement qualification set solutions, (2) information about design object description solutions, (3) information about the status of the design process, (4) information about the consumption of design resources, and (5) application specific design process results information.

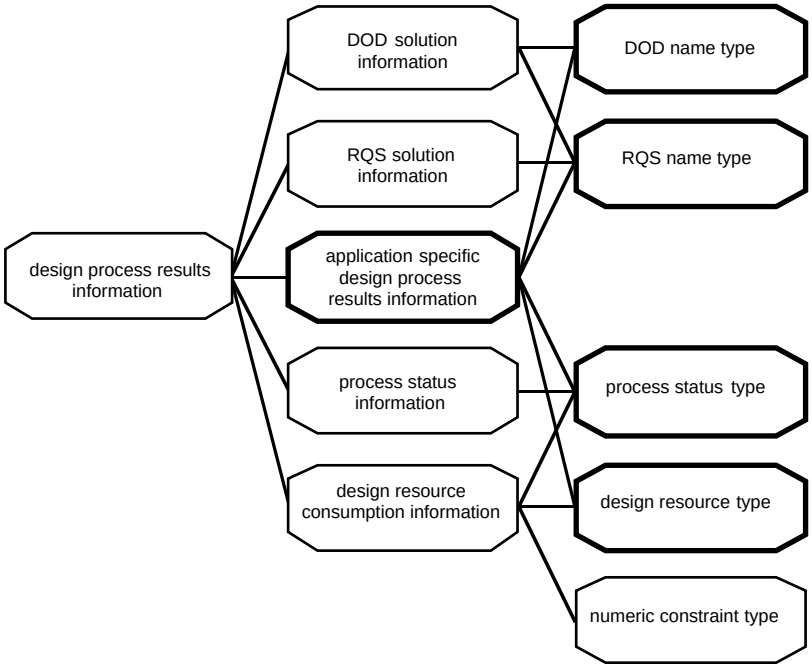


FIGURE 6.26. Composition of information type design process results information.

Figure 6.27 shows the structure of the information type RQS solution information, which models information about the requirement qualification set(s) generated as part of a solution by the design process. It contains two relations, is-acceptable-substitute-for and is-RQS-solution-to, which both have two arguments of the sort RQS-name.

An atom is-acceptable-substitute-for(*RQS-name1*, *RQS-name2*) specifies that the client of the design process commits to the requirement qualification set named *RQS-name1* as a substitute for the requirement qualification set named *RQS-name2*, meaning that the client of the design process accepts *RQS-name1* as an end result of the design process substituting *RQS-name2*. An atom is-RQS-solution-to(*RQS-name1*, *RQS-name2*) specifies that the requirement qualification set named *RQS-name1* is a solution to the requirement qualification set named *RQS-name2*, meaning that *RQS-name1* can be fulfilled and is consistent, unambiguous, precise, and complete, and so far in the design process, the client commits to *RQS-name1* as an acceptable substitute for *RQS-name2*.

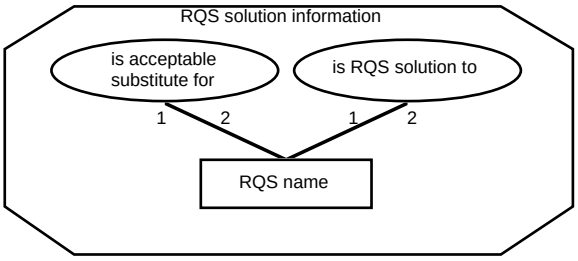


FIGURE 6.27. Structure of information type RQS solution information.

Example 6.15.
"The set of design requirements named RQS3 is an acceptable substitute for the set RQS2, and it is also a solution to the set RQS2."

is-acceptable-substitute-for(*RQS3*, *RQS2*)
is-RQS-solution-to(*RQS3*, *RQS2*)

Figure 6.28 shows the structure of the information type DOD solution information, which models information about the design object description(s) generated as part of a solution by the design process. It contains two relations, is-basis-reduct-of and is-DOD-solution-to.

The relation is-basis-reduct-of has two arguments that are both of the sort DOD-name. An atom is-basis-reduct-of(*DOD-name1*, *DOD-name2*) specifies that the design object description named *DOD-name1* is BASIS-reduct of the design object description named *DOD-name2*, meaning that *DOD-name1* only includes domain object information that is necessary for the construction of the design object.

The relation is-DOD-solution-to has two arguments of the sorts DOD-name and RQS-name, respectively. An atom is-DOD-solution-to(*DOD-name1*, *RQS-name1*) specifies that the design object description named *DOD-name1* is a solution to the requirement qualification set named *RQS-name1*, meaning that (1) *DOD-name1* is consistent, (2) restricted to the included domain object information necessary for the construction of the design object, it fulfils *RQS-name1*.

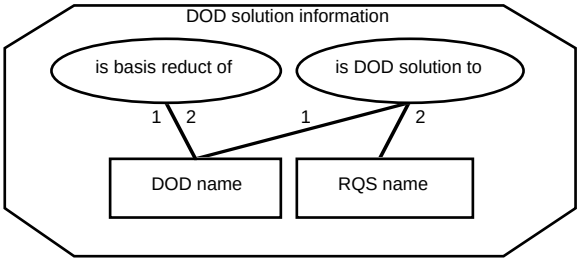


FIGURE 6.28. Structure of information type DOD solution information.

Example 6.16.
"The design object description named DOD43 is a basis reduct of the design object description named DOD42 and a solution to the set of design requirements named RQS3."

is-basis-reduct-of(*DOD43*, *DOD42*)
is-DOD-solution-to(*DOD43*, *RQS3*)

The information type process status type models the status of a design process. It contains the sort process status, which includes two objects: the object idle specifies that the process of concern is idle (and therefore not active), and the object active specifies that the process of concern is active (and therefore not idle).

Figure 6.29 shows the structure of the information type process status information, which models information about the status of a process. It contains three relations, is-previous-status, is-current-status, and is-next-status, which each have one argument of the sort process-status.

An atom is-previous-status(*process-status1*) specifies that the status of the concerned design process in its preceding state is *process-status1*. An atom is-current-status(*process-status1*) specifies that the status of the concerned design process in its current state is *process-status1*. An atom is-next-status(*process-status1*) specifies that the status of the concerned design process in its succeeding state is *process-status1*.

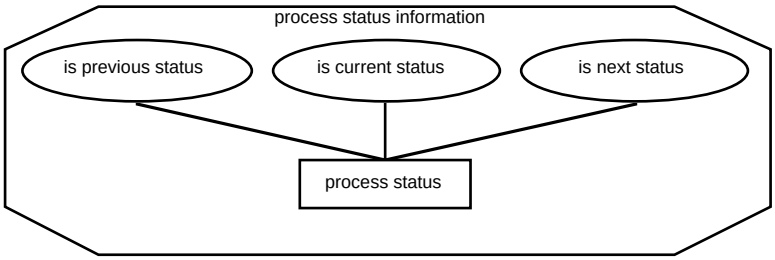


FIGURE 6.29. Structure of information type process status information.

Example 6.17.
"The design process has just started."

is-previous-status(idle)
is-current-status(active)

The information type design resource type models design resources that are needed to carry out a design process, such as a team of designers, a financial budget, availability of computers, and a budget of working hours. In practice, the use of design resources is most often an inevitable aspect of design processes.

Figure 6.30 shows the structure of the information type design resource consumption information, which models information about the consumption of design resources by a design process. It contains two relations, has-past-consumption-of and has-future-consumption-of, which each have two arguments of the sorts design-resource and numeric-constraint, respectively.

An atom has-past-consumption-of(*design-resource1*, *numerical-constraint1*) specifies that the design process has already used up *numerical-constraint1* units of the design resource *design-resource1*. An atom has-future-consumption-of(*design-resource1*, *numerical-constraint1*) specifies that the design process is expected to further use up *numerical-constraint1* units of the design resource *design-resource1*.

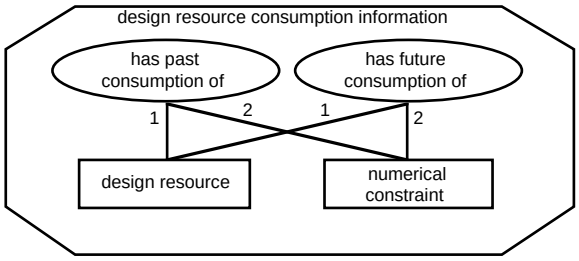


FIGURE 6.30. Structure of information type design resource consumption information.

Example 6.18.
"The designer reports to have worked 64 hours on the design project and to have used 2940 CPU seconds on the mainframe computer. She expects that completion of the design project will take at least 30 hours and at most 1500 CPU seconds."

has-past-consumption-of(*working-hours*, exactly(64))
has-past-consumption-of(*CPU-seconds*, exactly(2940))
has-future-consumption-of(*working-hours*, at-least(30))
has-future-consumption-of(*CPU-seconds*, at-most(1500))

The information type application specific design process results information models application specific information about the results of a design process. For example, this may include a rationale of the past and future consumption of design resources.

6.2.2.4 Structure and composition of knowledge at the third meta-level

The third and highest meta-level includes the following main information types: (1) design process objectives, (2) overall design strategies, (3) information about the strategic results of the processes of manipulating requirement qualification sets and design object descriptions, and (4) evaluations of the design process.

Design process objectives. Figure 6.31 shows the composition of the information type design process objectives, which models information about the design process objectives of a design process. The information type design process objectives refers to three other information types: (1) design process objective definitions, which models definitions of design process objectives, (2) design process objective enactment information, which models information about which design process objectives must actually be satisfied in a given situation, and (3) application specific design process objective information, which models application specific information about design process objectives.

The information type design process objective definitions refers to two information types: process objective definitions and qualified process objective definitions. These information types model the definitions of process objectives and qualified process objectives, respectively. Together, process objectives and qualified process objectives are termed *design process objectives*, which are modelled by the information type design process objective type.

A *process objective* is a requirement of a design process such as a deadline to be met or a limit on the consumption of a design resource. The information type process objective type is used to model process objectives. A process objective may be formulated directly in terms of a design process results information formula (modelled by an object of the sub-sort design-process-results-info-formula) or indirectly by means of a process objective name (modelled by an object of the sub-sort process-objective-name).

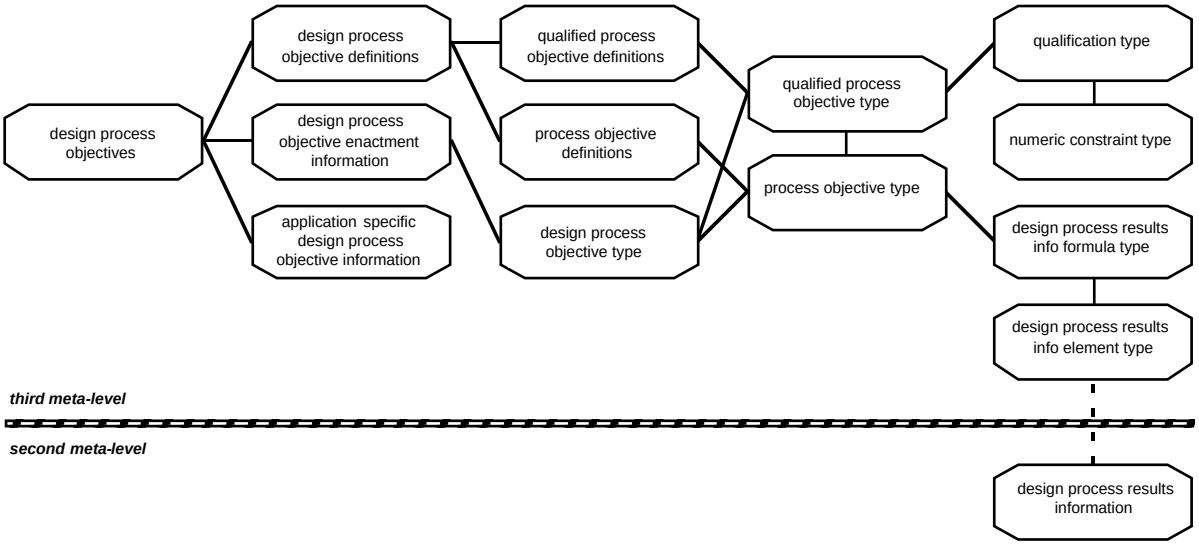


FIGURE 6.31. Composition of information type design process objectives.

The information type design process results info element type models meta-descriptions of design process results information. These meta-descriptions are obtained by the upward reflection of design process results information (modelled by the information type design proc-

ess results information), and are used to formulate epistemic information about design process results information.

A *design process results information formula* is expressed in the design process results language, which is used to describe the results of a design process and their relations. A design process results informatoin formula is composed of a finite number of design process results information elements (modelled by objects of the sort design-process-results-info-element) and the following functions for different predicate-logical operators:

- l-not for negation ($\neg$),
- l-or for disjunction ($\vee$),
- l-and for conjunction ($\wedge$), and
- l-implies for implication ($\Rightarrow$).

A *process objective name* is used to refer to a process objective; it is application specific and assumed to be unique within the context of a design process. Information about the design process results information formula corresponding to a given process objective name is modelled by the information type process objective definitions.

Figure 6.32 shows the structure of the information type process objective definitions. It contains the relation is-defined-as, which has two arguments of the sorts process-objective-name and design-process-results-info-formula, respectively. An atom is-defined-as(*process-objective-name1*, *design-process-results-info-formula1*) specifies that the process objective named *process-objective-name1* is defined as the design process results information formula *design-process-results-info-formula1* about the results of a design process and their relations.

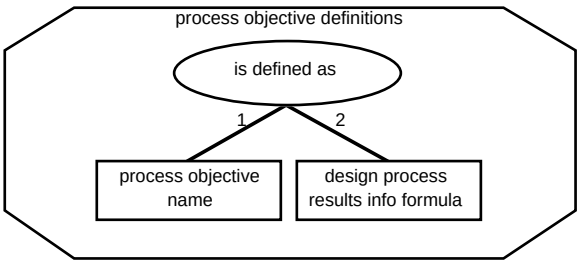


FIGURE 6.32. Structure of information type process objective definitions.

Example 6.19.
"The design process must be completed within 160 hours (PO1), preferably before July 28, 2002, at 9:00 a.m. (PO2), but at the latest on August 1, 2002, at 12:00 a.m. (PO3)."

is-defined-as(*PO1*, l-implies(l-and(is-previous-status(active), is-current-status(idle)),
has-past-consumption-of(*working-hours*, at-most(160))))

is-defined-as(*PO2*, l-implies(l-and(is-previous-status(active), is-current-status(idle)),
is-at-the-latest("2002-07-28, 09:00:00")))
is-defined-as(*PO3*, l-implies(l-and(is-previous-status(active), is-current-status(idle)),
is-at-the-latest("2002-08-01, 00:00:00")))

A *qualified process objective* specifies the (relative or absolute) importance of satisfying a list of given process objectives. Objects of the sort qualified-process-objective within the information type qualified process objective type are used to model qualified process objectives. A qualified process objective may be formulated directly as a qualified process objective expression (modelled by an object of the sub-sort qualified-process-objective-expression) or indirectly by means of a qualified process objective name (modelled by an object of the sub-sort qualified-process-objective-name).

A *qualified process objective expression* is composed of a qualification (modelled by an object of the earlier introduced sort qualification) and a process objective list (modelled by an object of the sort process-objective-list). This composition is specified by the function expr, which maps a pair of objects from the sort qualification and the sort process-objective-list on an object of the sort qualified-process-objective-expression.

A *qualified process objective name* is used to refer to a qualified process objective; it is application specific and assumed to be unique within the context of a design process. Information about the qualified process objective expression corresponding to a given qualified process objective name is modelled by the information type qualified process objective definitions.

Figure 6.33 shows the structure of the information type qualified process objective definitions. It contains the relation is-defined-as, which has two arguments of the sorts qualified-process-objective-name and qualified-process-objective-expression, respectively. An atom is-defined-as(*qualified-process-objective-name1*, *qualified-process-objective-expression1*) specifies that the qualified process objective named *qualified-process-objective-name1* is defined as the qualified process objective expression *qualified-process-objective-expression1*.

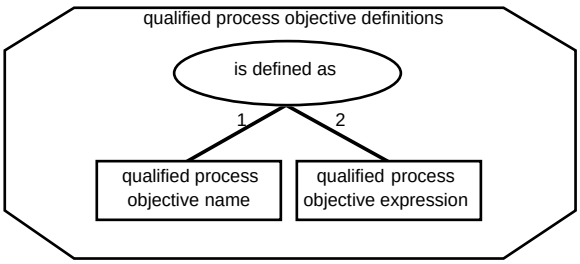


FIGURE 6.33. Structure of information type qualified process objective definitions.

Example 6.20.
"Process objective PO1 must be satisfied (QPO1). Either process objective PO2 or process objective PO3 must be satisfied, which one does not matter (QPO2)."

is-defined-as(*QPO1*, expr(every, [*PO1*]))
is-defined-as(*QPO2*, expr(at-random(any), [*PO2*, *PO3*]))

Figure 6.34 shows the structure of the information type design process objective enactment information, which models information about which design process objectives must actually be satisfied in a given situation. It contains the relation is-to-be-satisfied, which has one argument of the sort design-process-objective. An atom is-to-be-satisfied(*design-process-objective1*) specifies that design process objective *design-process-objective1* is to be satisfied.

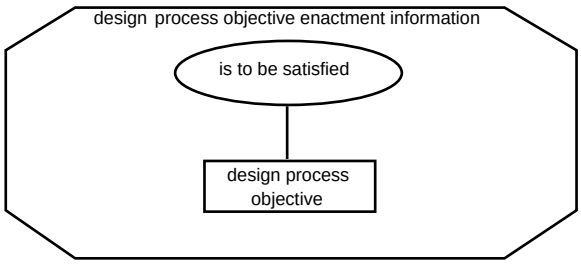


FIGURE 6.34. Composition of information type design process objective enactment information.

Example 6.21.
"The process objective PO1 and the qualified process objective QPO2 must be satisfied, as well as the (unnamed) qualified process objective that process objective PO4 should be satisfied, if possible."

is-to-be-satisfied(*PO1*)
is-to-be-satisfied(*QPO2*)
is-to-be-satisfied(expr(at-random(all-possible), [*PO4*]))

The information type application specific design process objective information models application specific information about design process objectives. For example, in some types of design processes different design parties are involved, such as the customer and the designer. In such situations, it may be of interest to know which design party is the source of a specific design process objective. This information can be modelled by means of a relation *is-source-of* with two arguments of the sorts *design-party* and design-process-objective, respectively, which is specified within the information type application specific design process objective information.

Overall design strategy. Figure 6.35 shows the composition of the information type overall design strategy, which models information about overall design strategies.

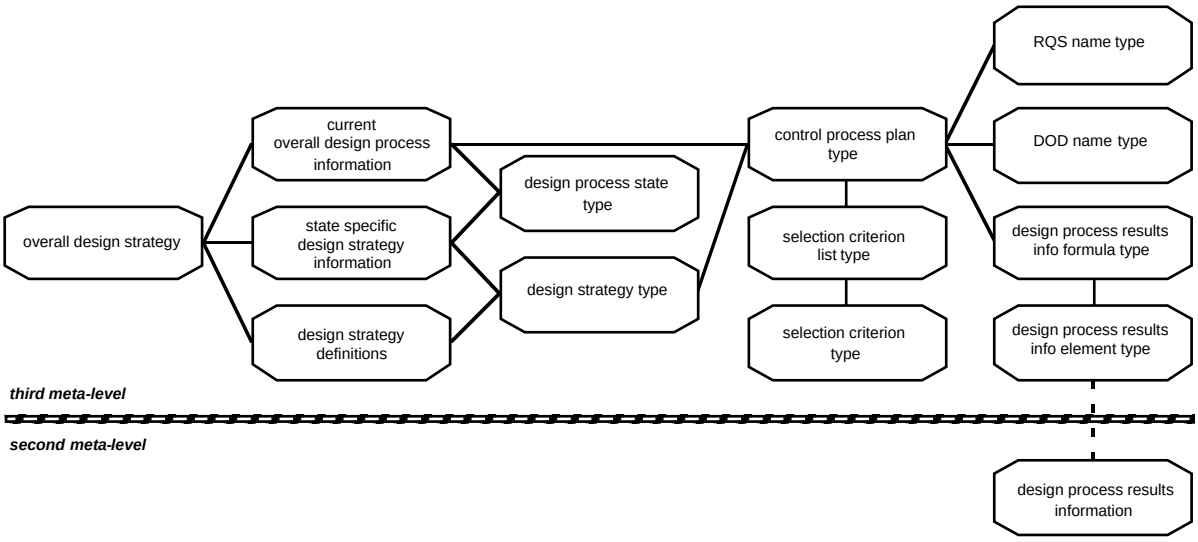


FIGURE 6.35. Composition of information type overall design strategy.

The information type overall design strategy refers to three information types: (1) current overall design process information, which models information about the identity of the current overall design process state and the contents of the current control process plan, (2) state specific overall design strategy information, which models information about the overall design strategies included by design process states, and (3) design strategy definitions, which models the definitions of design strategies.

A *design process state* is an information state of a design process. Objects of the sort design process state within the information type design process state type are used to model design process states. Each state is assumed to be uniquely identifiable within a design process.

A *design strategy* specifies what the behaviour of the design process should be, or which results it should have achieved in its next state. Objects of the sort design strategy within the information type design strategy type are used to model design strategies. A design strategy may be formulated directly in terms of a control process plan (modelled by an object of the sub-sort control-process-plan) or indirectly by means of a design strategy name (modelled by an object of the sub-sort design-strategy-name).

The information type control process plan type models plans to guide a design process. A primitive control process plan is built up from requirement qualification set names, design object description names, design process results information formulae, and selection criterion lists. A complex control process plan is built from other control process plans, using ordering principles such as repetition, sequencing and parallelism.

A *selection criterion* is used for the selection of requirement qualification sets, design object descriptions, or elements of such sets or descriptions, which are subject to modification or retrieval. Objects of the sort selection-criterion-list within the information type selection criterion list type are used to model lists of (application specific) selection criteria.

A *control approach* is a particular approach to control the modification of the current requirement qualification or current design object description. Objects of the sort control-approach within the information type control approach type are used to model different control approaches. The following control approaches are based on Treur's (non-exhaustive) list of specific uses of reasoning about design requirements [Treur, 1991] and can be applied to generate modifications to both requirement qualification sets and design object descriptions:

- a *transformation* of the available information into different but equivalent information (e.g., transformation of the equation $x^2 + 4x - 5 = 0$ into $(x - 1)(x + 5) = 0$);
- a *translation* of the available information into another language (e.g., translation of the equation $(x - 1)(x + 5) = 0$ into $(y - 3)(y + 3) = 0$, plus the extra equation $y = x + 2$);
- a *decomposition* of the available information into new information (e.g., decomposition of global requirements of a software system into more detailed requirements);
- a *composition* of new information from the available information (e.g., composition of a software system from a set of sub-systems and a set of sub-system interfaces);
- a *reduction* of the available information (e.g., removing a design requirement);
- an *extension* of the available information (e.g., adding a design requirement).

These control approaches have been used and observed in different applications. The following functions are used to model control process plans (such as the ones introduced in Chapter 12 of this thesis):

- continue-with(*RQS-name1*, *DOD-name1*) models the control process plan that the design process has to continue to determine a solution on the basis of the requirement qualification set named *RQS-name1* and the design object description named *DOD-name1*;
- if-then(*design-process-results-info-formula1*, *control-process-plan1*) models the control process plan that if *design-process-results-info-formula1* holds, then the control process plan *control-process-plan1* is executed;
- if-then-else(*design-process-results-info-formula1*, *control-process-plan1*, *control-process-plan2*) models the control process plan that if *design-process-results-info-formula1* holds, then the control process plan *control-process-plan1* is executed, and otherwise the control process plan *control-process-plan2*;
- while-do(*design-process-results-info-formula1*, *control-process-plan1*) models the control process plan that, as long as *design-process-results-info-formula1* holds, the control process plan *control-process-plan1* is executed;

- repeat-until(*control-process-plan1*, *design-process-results-info-formula1*) models the control process plan that the control process plan *control-process-plan1* is executed until *design-process-results-info-formula1* holds;
- do-in-sequence(*control-process-plan1*, *control-process-plan2*) models the control process plan that first the control process plan *control-process-plan1* is executed and then the control process plan *control-process-plan2*;
- do-in-parallel(*control-process-plan1*, *control-process-plan2*) models the control process plan that the control process plan *control-process-plan1* is executed at the same time as the control process plan *control-process-plan2*;
- apply-criteria-for-retrieval-approach(*selection-criterion-list1*, *control-approach1*) models the control process plan that the control approach *control-approach1* is applied to the current requirement qualification set (design object description), on the basis of requirement qualification sets (design object descriptions) retrieved from the design history that meet the selection criteria from the list *selection-criterion-list1*;
- apply-criteria-for-modification-approach(*selection-criterion-list1*, *control-approach1*) models the control process plan that the control approach *control-approach1* is applied to the current requirement qualification set (design object description), of which those elements are modified that meet the selection criteria from the list *selection-criterion-list1*.

Figure 6.36 shows the structure of the information type current overall design process information, which models information about the identity of the current state of an overall design process and the contents of the current control process plan. It contains two relations, is-current-overall-design-process-state and is-current-control-process-plan.

The relation is-current-overall-design-process-state has one argument of the sort design-process-state. An atom is-current-overall-design-process-state(*design-process-state1*) specifies that the current state of the overall design process is *design-process-state1*. The relation is-current-control-process-plan has one argument of the sort control-process-plan. An atom is-current-control-process-plan(*control-process-plan1*) specifies that the current control process plan of the overall design process is *control-process-plan1*.

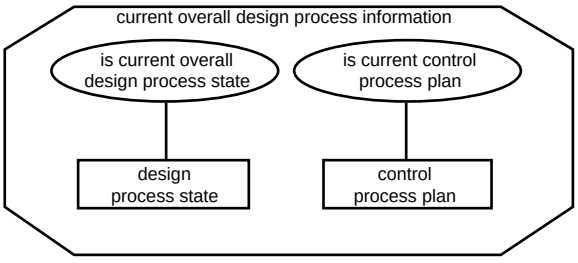


FIGURE 6.36. Structure of information type current overall design process information.

Example 6.22.
"The current state of the overall design process is named State198. The current control plan is to continue with requirement qualification set RQS123 and design object description DOD453."

is-current-overall-design-process-state(*State198*)
is-current-control-process-plan(continue-with(*RQS123*, *DOD453*))

Figure 6.37 shows the structure of the information type state specific design strategy information, which models information about the design strategies involved in specific design process states. It contains the relation includes-design-strategy, which has two arguments of the sorts design-process-state and design-strategy, respectively. An atom includes-design-strategy(*design-process-state1*, *design-strategy1*) specifies that the design process state *design-process-state1* involves the design strategy *design-strategy1*.

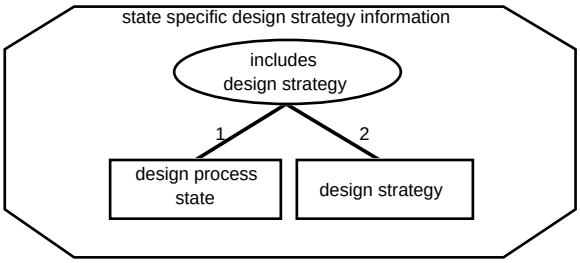


FIGURE 6.37. Structure of information type state specific design strategy information.

Example 6.23.
"The state of the overall design process designated State198 includes a strategy to pursue an explorative approach. The state of the requirement qualification set manipulation process designated RQSMState20 includes a strategy to renegotiate the initial design requirements that are in conflict with each other."

includes-design-strategy(*State198*, *explorative-approach*)
includes-design-strategy(*RQSMState20*, *renegotiate-initial-conflicting-design-requirements*)

Figure 6.38 shows the structure of the information type design strategy definitions, which models definitions of design strategies. It contains the relation is-defined-as, which has two arguments of the sorts design-strategy-name and control-process-plan, respectively. An atom is-defined-as(*design-strategy-name1*, *control-process-plan1*) specifies that the design strategy named *design-strategy-name1* is defined as the control process plan *control-process-plan1*.

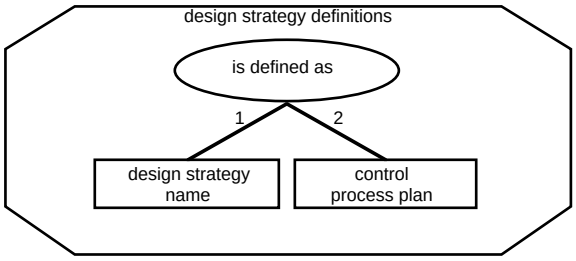


FIGURE 6.38. Structure of information type design strategy definitions.

Example 6.24.
"The strategy to renegotiate the initial design requirements that are in conflict with each other is defined as the following control process plan: delete those design requirements from the current requirement qualification set that have been introduced at the start of the design process and that are inconsistent with each other."

is-defined-as(*renegotiate-initial-conflicting-design-requirements*,
apply-criteria-for-modification-approach(
[*is-introduced-at-start*, *is-part-of-inconsistency*], reduction))

Control process evaluations. Figure 6.39 shows the composition of the information type control process evaluations, which models evaluations of a requirement qualification set manipulation process and a design object description manipulation process with respect to specific overall design strategies. These control process evaluations are used by design process co-ordination to determine whether or not to continue the design process.

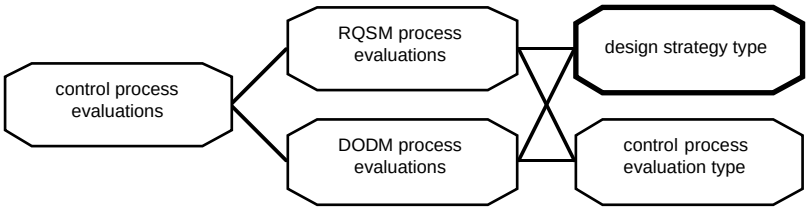


FIGURE 6.39. Composition of information type control process evaluations.

The information type control process evaluation type models strategic evaluations of control processes. The sort control-process-evaluation contains the objects incomplete, succeeded and failed. Given an overall design strategy, the object incomplete specifies that a control process has not been able to complete the strategy (due to a lack of information), the object succeeded specifies that a control process has completed the strategy with success, and the object failed specifies that a control process has completed the strategy with failure.

Figure 6.40 shows the structure of the information type DODM process evaluations, which models evaluations of a design object description manipulation process in relation to overall design strategies. It contains the relation has-DODM-process-evaluation, which has two arguments of the sorts design-strategy and control-process-evaluation, respectively. An atom has-DODM-process-evaluation(*design-strategy1*, *control-process-evaluation1*) specifies that the attempt by the design object description manipulation process to complete the design strategy *design-strategy1* has *control-process-evaluation1* as its result.

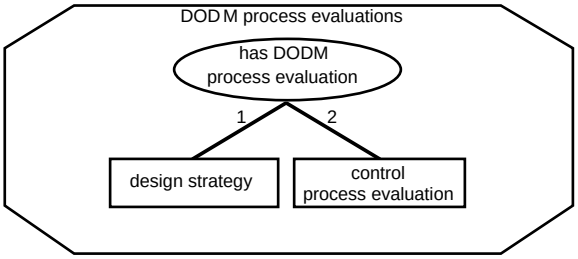


FIGURE 6.40. Structure of information type DODM process evaluations.

Example 6.25.
"Overall design strategy ODS1 has been completed with failure as a result."

has-DODM-process-evaluation(*ODS1*, failed)

Figure 6.41 shows the structure of the information type RQSM process evaluations, which models evaluations of a requirement qualification set manipulation process in relation to overall design strategies. It contains the relation has-RQSM-process-evaluation, which has two arguments of the sorts design-strategy and control-process-evaluation, respectively. An atom has-RQSM-process-evaluation(*design-strategy1*, *control-process-evaluation1*) specifies that the attempt by the requirement qualification set manipulation process to complete the design strategy *design-strategy1* has *control-process-evaluation1* as its result.

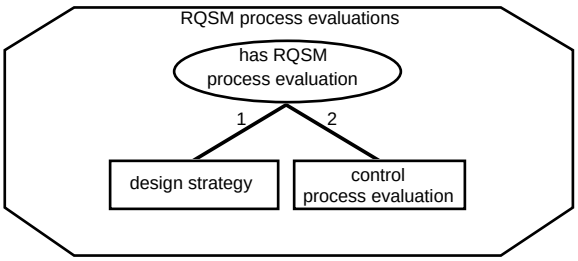


FIGURE 6.41. Structure of information type RQSM process evaluations.

Example 6.26.
"Overall design strategy ODS1 has been completed with success as a result."

has-RQSM-process-evaluation(*ODS1*, succeeded)

Design process evaluations. Figure 6.42 shows the composition of the information type design process evaluations, which models evaluations of a design process concerning its performance and in relation to given design process objectives.

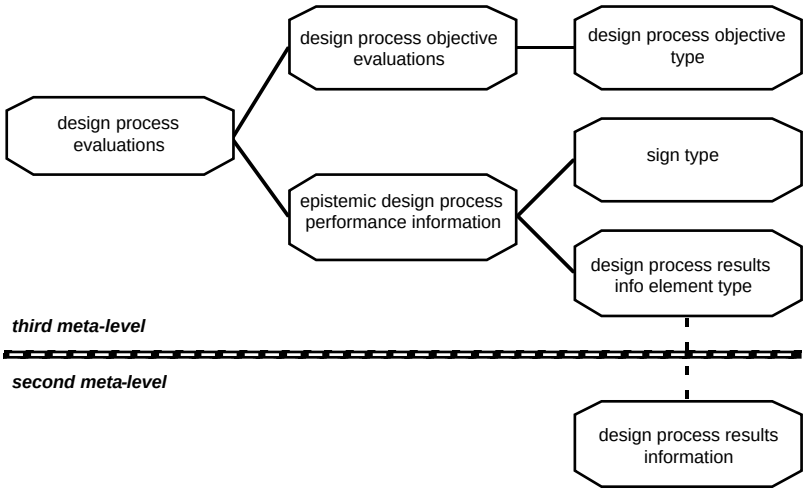


FIGURE 6.42. Composition of information type design process evaluations.

Figure 6.43 shows the structure of the information type epistemic design process performance information, which models epistemic information about the performance indicators of a specific design process. It contains the relation is-part-of-design-process-results, which has two arguments of the sort design-process-results-info-element and sign, respectively. An atom is-part-of-design-process-results(*design-process-results-info-element1*, *sign1*) specifies that results of the design process include, among others, the information *design-process-results-info-element1*, with sign *sign1* as its truth value.

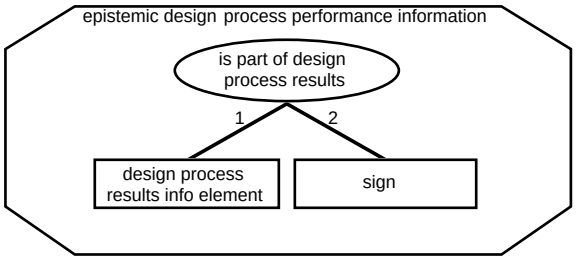


FIGURE 6.43. Structure of information type epistemic design process performance information.

Example 6.27.
"The design process has been completed on July 31, 1999, at 8:15 p.m., but it is not known whether the design process has been completed within 160 hours or not."

is-part-of-design-process-results(is-previous-state(active), pos)
is-part-of-design-process-results(is-current-state(idle), pos)
is-part-of-design-process-results(*is-current-time*("1999-07-31, 20:15:00"), pos)
is-part-of-design-process-results(has-past-consumption-of(*working-hours*, at-most(160)), unk)

Figure 6.44 shows the structure of the information type design process objective evaluations, which models evaluations of the design process objectives of a design process. It contains three relations, is-satisfied, is-violated and is-decided, which each have one argument of the sort design-process-objective.

An atom is-satisfied(*design-process-objective1*) specifies that the design process satisfies the design process objective *design-process-objective1*. An atom is-violated(*design-process-objective1*) specifies that the design process violates the design process objective *design-process-objective1*. An atom is-decided(*design-process-objective1*) specifies that it is known that the design process either satisfies or violates the design process objective *design-process-objective1*. The negation of this atom is used to specify that it is not known whether the design process satisfies or violates the design process objective *design-process-objective1*.

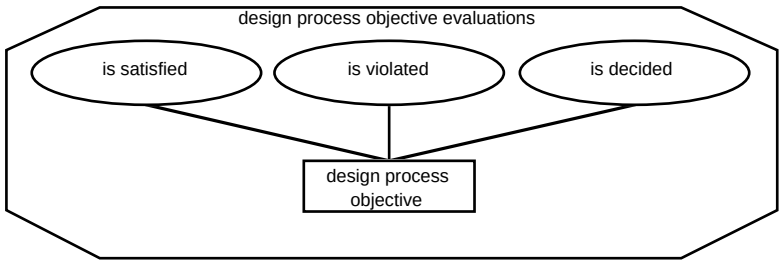


FIGURE 6.44. Structure of information type design process objective evaluations.

Example 6.28.
"The design project satisfies process objectives PO1 and PO3 and qualified process objective QO1, but it violates process objective PO2."

is-satisfied(*PO1*)
is-satisfied(*PO3*)
is-satisfied(*QPO1*)
is-violated(*PO2*)

6.3 Relation between Compositions of Process and Knowledge

The relation between the process composition and the knowledge composition of a design process specifies how the four processes involved in design relate to the four reflection levels for a design process.

- The object level is hidden inside DOD manipulation, and contains domain object information.

- The first meta-level is the lowest visible level for design as well as for DOD manipulation, and it is hidden inside RQS manipulation. It contains design object descriptions and design requirements.

- The second meta-level is the lowest visible level for RQS manipulation. It contains requirement qualification sets, DOD assessments, and RQS assessments, as well as proposals for modification of design object descriptions and proposals for modification of requirement qualification sets.

- The third meta-level is the only visible level for design process co-ordination and it is the highest visible level for design and for RQS manipulation as well as DOD manipulation. It contains design process objectives, overall design strategies, manipulation process evaluations and design process evaluations.

Figure 6.45 shows how the relation between process composition and knowledge composition of a design process is modelled in GDM.

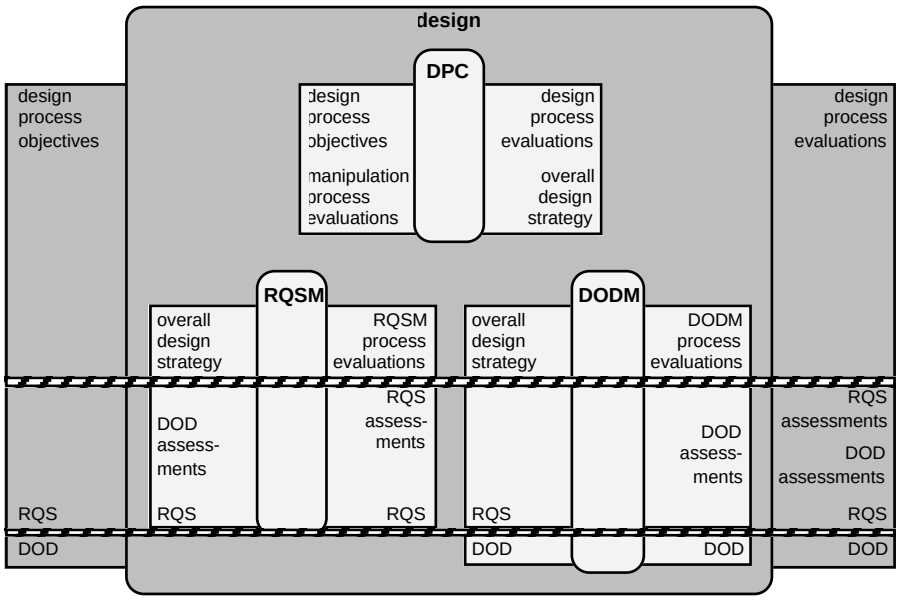


FIGURE 6.45. Relation between process composition and knowledge composition of a design process.

6.4 Use of the Model in Analysis and Development

To be able to analyse practical design processes, and for the development of more advanced design support systems that provide human designers with real support for reasoning about various aspects of design, it is necessary to have a thorough understanding of design processes. For this purpose, the generic design model GDM has been developed.

GDM is a blueprint of the generic features of design processes. That is, GDM models the significant types of information and knowledge that play a role within a design process, irrespective of the specific design method and application domain. By using GDM to model a specific type of design process, a modeller can focus directly on the application specific aspects of the design process (e.g., the design method and the domain of application). The modeller may decide to refine GDM by including application specific sub-components (and therefore also new information links and task control involving these components), information types, and knowledge bases.

This section explains how the model of the two highest levels of a design process that GDM provides can be used in the analysis of practical design processes and the development of design support systems.

Identification of design processes. During the analysis of applications, the following checklist can be used to determine whether or not a given application specific process is a design process.

- The input of the process consists at least of a set of design requirements, where each design requirement specifies a condition on some object, regarding its behaviour, function, form and/or structure.
- The output of the process consists at least of a description of some object in terms of behaviour, function, form and/or structure.
- The intent of the process is to produce a description of an object that satisfies the given set of design requirements.

Application specific process composition. For any application specific design process, the sub-processes involved can be modelled by the components DPC, RQSM, and DODM. Chapter 7 describes the generic process composition of requirement qualification set manipulation processes, which are modelled by sub-components of RQSM. Chapter 8 describes the generic process composition of design object description manipulation processes, which are modelled by sub-components of DODM.

A modeller may decide to refine the component DPC to model the application specific generation and evaluation of overall design strategies. This can be done in two ways:

- by refinement of the process composition, that is, by modelling the sub-processes involved in an application specific design process co-ordination process;
- by refinement of the knowledge composition, that is, by modelling the design process co-ordination knowledge involved in an application specific design process co-ordination process.

Application specific knowledge composition. The following information types within GDM can be used to model application specific objects and relations. (Note that in the detailed textual specification of GDM in Appendix B, these information types either have names that contain application specific as a prefix, or they include objects of which the names contain Example as a prefix.)

- The information type domain object type can be used to model constants and variables that apply to a specific domain of application, by means of objects of the sort domain-object-constant (for application specific constants) and objects of the sort domain-object-variable (for application specific variables).
- The information type attribute type can be used to model attributes of objects within a specific domain of application, by means of objects of the sort attribute.
- The information type value type can be used to model attribute values of objects within a specific domain of application, by means of objects of the sort value.
- The information type application specific domain object information can be used to model application specific information about objects within a specific domain of application, by means of relations on the sort domain-object.

- The information type DOD name type can be used to model names of design object descriptions, by means of objects of the sort DOD-name.
- The information type requirement type can be used to model names of requirements, by means of objects of the sort requirement-name.
- The information type qualified requirement type can be used to model names of qualified requirements, by means of objects of the sort qualified-requirement-name.
- The information type application specific design requirement information can be used to model application specific information about design requirements, by means of relations on the sort design-requirement.
- The information type RQS name type can be used to model names of requirement qualification sets, by means of objects of the sort RQS-name.
- The information type design resource type can be used to model design resources (such as a financial budget, manpower, etc.), by means of objects of the sort design-resource.
- The information type application specific design process results information can be used to model application specific information about the results of a design process, by means of relations on the sorts DOD-name, RQS-name, process-status, and design-resource.
- The information type process objective type can be used to model names of process objectives, by means of objects of the sort process-objective-name.
- The information type qualified process objective type can be used to model names of qualified process objectives, by means of objects of the sort qualified-process-objective-name.
- The information type application specific design process objective information can be used to model application specific information about design process objectives, by means of relations on the sort design-process-objective.
- The information type selection criterion type can be used to model selection criteria for use in the modification or retrieval of requirement qualification sets and design object descriptions, by means of objects of the sort selection-criterion.
- The information type design strategy type can be used to model names of design strategies, by means of objects of the sort design-strategy-name.
- The information type design process state type can be used to model states of a design process, by means of objects of the sort design-process-state.

Chapter 7

Requirement Qualification Set Manipulation

The design requirements given at the beginning of a design process are most often inconsistent, ambiguous, vague, or incomplete. The requirement qualification set manipulation process is responsible for generating, modifying and deductively refining requirement qualification sets. This chapter describes the model of requirement qualification set manipulation processes that GDM provides, which details the third level of process abstraction of a design process.

An essential aspect of design processes is the formulation of design requirements that properly describe the desired or needed behaviour, function, form or structure of a design object. A requirement qualification set, which is a set of such design requirements, is the most characteristic input of a design process.

In practical situations, it may be unnecessary or impossible to satisfy all initial requirements placed on a design object. Although a requirement is always meant to be satisfied, sometimes it is acceptable if one of the requirements is violated, especially if it means that at the same time one or more other requirements can be satisfied. In a design process, qualified requirements are used to express the relative or absolute importance of satisfying specific requirements. If all qualified requirements included in a given requirement qualification set are satisfied (and not necessarily all requirements included in the set), then this set is fulfilled. But even with qualified requirements, it may be impossible to fulfil the initially given set.

During a design process, it may be established that the given requirement qualification set is ill structured, which renders it is useless to try generating a satisfactory design object description for that set. It has been earlier explained (following the argumentation of Smithers, Corne and Ross [Smithers, Corne and Ross, 1994]) that an ill structured set is one that is:

- *inconsistent* (i.e., two or more of the design requirements included in the set cannot be satisfied by the same design object description),
- *ambiguous* (i.e., there exist design object descriptions that fulfil the set and that are inconsistent with each other),
- *imprecise* (i.e., at least one of the design requirements included in the set cannot be decided to be satisfiable), or
- *incomplete* (i.e., according to a design requirements theory of the application domain, there are design requirements missing in the set).

The ill-structuredness of a requirement qualification set may be inevitable for several reasons. For example, it may be difficult to recognise inconsistencies between the initial design requirements. This may have to do with the sheer size of the initial requirement qualification set, but it may also be more or less inherent to the domain of application. In mechanical engineering, for instance, the initial requirements are usually well structured, but in architecture, the initial requirements are often imprecise and need further refinement before any inconsistency will come to the surface.

During a design process, a *requirement qualification set manipulation process* modifies and deductively refines the initial requirement qualification set, with the goal of delivering a well defined (i.e., consistent, unambiguous, precise, and complete) set of design requirements. Together with a design process co-ordination process and a design object description manipulation process, a requirement qualification set manipulation process forms a subprocess of a design process.

In Chapter 6, the two highest process abstraction levels of a design process have been described. This chapter describes the model of requirement qualification set manipulation processes that GDM provides, which details the third level of process abstraction of a design process. Section 7.1 presents the process composition of a requirement qualification set manipulation process, Section 7.2 the knowledge composition of a requirement qualification set manipulation process, and Section 7.3 the relation between process composition and knowledge composition of a requirement qualification set manipulation process. Finally, Section 7.4 explains how the model of a requirement qualification set manipulation process that GDM provides can be used in the analysis of practical design processes and the development of design support systems.

7.1 Process Composition

This section describes processes identified in requirement qualification set manipulation, and the composition of these processes.

7.1.1 Processes at different abstraction levels

This sub-section describes processes involved in requirement qualification set manipulation and the different levels of abstraction at which these processes play a role.

7.1.1.1 Processes

The following processes are involved at the two highest process abstraction levels of a requirement qualification set manipulation process:

- requirement qualification set manipulation (*RQSM*),
- requirement qualification set modification (*RQS modification*),
- requirement qualification set manipulation history maintenance (*RQSM history maintenance*),
- current requirement qualification set maintenance (*current RQS maintenance*),
- deductive requirement qualification set refinement (*deductive RQS refinement*).

In the following, these five processes are explained, together with the type of input information they use and the types of output information they produce. Note that requirement qualification set manipulation processes have already been introduced in Chapter 6; for the reader's convenience, parts of the explanation are repeated, and the same bicycle design example is used.

Requirement qualification set manipulation
On the basis of a given requirement qualification set, and in interaction with stake-holders (such as a client), a requirement qualification set manipulation process aims to generate a well defined requirement qualification set that includes sufficient design requirement information for the generation of a satisfactory design object description. This process always operates on a (possibly partial) set of design requirements, called the *current requirement qualification set*.

The *current RQSM process state* defines the current state of the requirement qualification set manipulation process. Only by well defined operations (such as the modification or deductive refinement of the current requirement qualification set) can the process as a whole make a transition to a new state, which then becomes the current state.

In some applications, requirement qualification set manipulation has a simple role; for example, in a configuration process as described in Chapter 10, only individual requirements are replaced. In other applications, requirement qualification set manipulation has a more complex role; for example, in an aircraft re-design process as described in Chapter 13, requirement qualification set manipulation applies different methods to generate and modify requirement qualification sets, such as *extension-by-retrieval* and *reduction-by-modification.*

Figure 7.1 shows (on the left-hand side) the types of information that a requirement qualification set manipulation process uses as input and (on the right-hand side) the types of information that it produces as output.

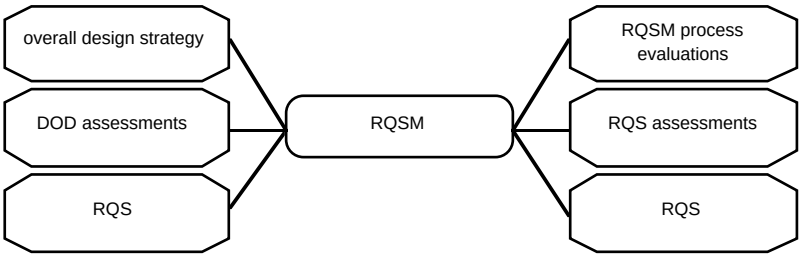


FIGURE 7.1. Input and output of a requirement qualification set manipulation process.

Refer to Chapter 6 for an example explaining the types of input and output information used by a requirement qualification set manipulation process.

Requirement qualification set modification

A requirement qualification set modification process aims to modify the current requirement qualification set into a well defined set that includes sufficient design requirement information for the generation of a satisfactory design object description. This process acts on the basis of information about present and past states of the manipulation process, about steps (i.e., state transitions) taken within the manipulation process, about traces (i.e., sequences of steps) generated within the manipulation process, and about the intermediate results of the manipulation process. The process may modify the contents of the current requirement qualification set, or set a focus for deductive refinement of the current requirement qualification set. The process may also decide to retrieve an earlier generated requirement qualification set (to replace the current set), inspect the history of the requirement qualification set manipulation process (by expressing queries to be processed), or terminate the manipulation process.

Figure 7.2 shows the types of information that a requirement qualification set modification process uses as input and the types of information that it produces as output.

FIGURE 7.2. Input and output of a requirement qualification set modification process.

The following example, extending the bicycle design example of Chapter 6, is used to explain the types of input information used by a requirement qualification set modification process.

Example 7.1.
"The requirements included in the current set (*Definition Study Doc v0.4*) are, among others, that the bicycle must be suited for riding in mountains and safe to be used by young children, and it must cost less than 400 euro. If the latter two requirements are irreconcilable, safety is preferred over price; the first requirement, though, must be satisfied. One design of a bicycle is available (*Design Doc v0.2*), which does not satisfy any of the requirements included in the set named *Definition Study Doc v0.3*.

In the design process, five requirement qualification set manipulation states have been generated so far. In the fourth state, the current requirement qualification set was *Definition Study Doc v0.3*, and the alteration proposal was selected to replace the design requirement that the bicycle be low-priced by a design requirement that the bicycle's price must be less than 400 euro. The current state is designated *State5*.

The control decision taken in the first state was to deductively refine the current set, in the second state to modify the current set, in the third state to deductively refine the current set, and in the fourth state to modify the current set. The overall design strategy since the first state has been to use earlier bicycle design cases as a basis. Up to now, there has been no evaluation of this overall design strategy."

RQS modification basis. The *RQS modification basis* of a requirement qualification set modification process concerns basis information for preparing the next step of the requirement qualification set manipulation process. In GDM, the information type RQS modification basis can be used to model the RQS modification basis, including the following types of information:

- information about the design requirement information included in specific requirement qualification sets,
- design object description assessments,
- requirement qualification set assessments,
- information about the identity of the current requirement qualification set,
- currently applicable proposals for alterations to the current requirement qualification set,
- currently applicable rejections of alterations to the current requirement qualification set,
- information about the composition of alterations to the current requirement qualification set,
- application specific information about alterations to the current requirement qualification set, and
- information about requirement qualification set solutions.

See Table 7.1 for the requirement qualification set modification basis in Example 7.1.

TABLE 7.1. RQS modification basis in Example 7.1.

Kind	*Content*
RQS	The requirements included in the set named *Definition Study Doc v0.4* are that the bicycle must be suited for riding in mountains and safe to be used by young children, and it must cost less than 400 euro. If the latter two requirements are irreconcilable, safety is preferred over price; the first requirement, though, must be satisfied.
DOD assessments	The design named *Design Doc v0.2*, does not satisfy any of the requirements included in the set *Definition Study Doc v0.3*.
Current RQS identity information	The current set is named *Definition Study Doc v0.4*.

RQSM trace information. A requirement qualification set modification process may need information about earlier states of the requirement qualification set manipulation process of which it is part, such as information about which alterations have already been tried, the sequence in which these were made, what the results were, and so forth. The term *RQSM trace information* denotes such information, which is modelled in GDM by the information type RQSM trace information. Three types of RQSM trace information are distinguished:

- design process state specific information about the modification of requirement qualification sets,
- generic information about the trace of the requirement qualification set manipulation process (e.g., information about the sequence of states in the manipulation process, the decisions for the steps made within the manipulation process, and the overall design strategy followed in a specific state), and
- application specific information about individual states of the requirement qualification set manipulation process and their relations.

See Table 7.2 for part of the RQSM trace information in Example 7.1.

TABLE 7.2. RQSM trace information in Example 7.1.

Kind	*Content*
State specific RQS modification process information	In the fourth state, the current requirement qualification set was *Definition Study Doc v0.3* and the proposal was selected to replace the design requirement that the bicycle be low priced by a design requirement that the bicycle's price must be less than 400 euro.
Manipulation trace information	There are five states so far. The decision in the first state was to deductively refine the current set, in the second state to modify the current set, in the third state to deductively refine the current set, and in the fourth state to modify the current set. The overall design strategy since the first state is to use earlier bicycle design cases as a basis. Up to now, there has been no evaluation of this strategy.

The following example, extending Example 7.1, is used to explain the types of output information produced by a requirement qualification set modification process.

Example 7.2.
"Two of the design requirements included in the set named *Definition Study Doc v0.4*, concerning the bicycle's suitability for riding in mountains and usability of the bicycle by young children, cannot be decided to be satisfiable. It is decided to derive structural integrity requirements (e.g., which weight the bicycle must be able to carry, which forces it must be able to endure, etc.)."

RQS modification results. With each step, a requirement qualification set modification process produces new (intermediate) results. The *RQS modification results* denote the modification information produced in the current requirement qualification set manipulation process state, which is modelled in GDM by the information type RQS modification results. The following types of RQS modification results are distinguished:

- assessments of requirement qualification sets,
- information about the identity of a requirement qualification set to be retrieved,
- proposed alterations to the current requirement qualification set,
- selected alterations to the current requirement qualification set,
- information about the composition of alterations to the current requirement qualification set,
- the design requirement information in focus for deductive refinement of the current requirement qualification set,
- queries for information about the contents of requirement qualification sets,
- application specific information about alterations to the current requirement qualification set, and
- information about requirement qualification set solutions.

These requirement qualification set modification results are optional as output of a requirement qualification set modification process. See Table 7.3 for the requirement qualification set modification results in Example 7.2.

TABLE 7.3. RQS modification results in Example 7.2.

Kind	*Content*
RQS assessments	Two design requirements included in the set named *Definition Study Doc v0.4* about the bicycle's suitability for riding in mountains and usability of the bicycle by young children, cannot be decided to be satisfiable.
Deductive RQS refinement focus	It is decided to derive structural integrity requirements (e.g., which weight the bicycle must be able to carry, which forces it must be able to endure, etc.).

RQSM step information. Besides information related to modification results, a requirement qualification set modification process produces process-related information about the next manipulation step to be taken. The *RQSM step information* of a requirement qualification set modification process denotes this type of information, which is modelled in GDM by the information type RQSM step information. Five types of step information are distinguished:

- information about the decision for the next step of the requirement qualification set manipulation process (i.e., to retrieve an earlier generated set, to modify the current set, to deductively refine the current set, to inspect the history of the manipulation process, or to terminate the manipulation process),
- requirement qualification set manipulation process evaluations,
- queries for generic, design process state specific, information about the modification of requirement qualification sets,
- queries for generic information about the trace of the requirement qualification set manipulation process, and
- queries for application specific information about individual states of the requirement qualification set manipulation process and their relations.

Except for the current control decision, these types of information are optional as output of a requirement qualification set modification process. See Table 7.4 for part of the requirement qualification set manipulation step information in Example 7.2.

TABLE 7.4. RQSM step information in Example 7.2.

Kind	*Content*
Current control decision information	It is decided to deductively refine the current set.

Requirement qualification set manipulation history maintenance

A requirement qualification set manipulation (RQSM) history maintenance process aims to record the steps and results of a requirement qualification set manipulation process and to provide historical information that is of use in the current state of the manipulation process.

RQSM history maintenance necessarily involves truth maintenance, to prevent the requirement qualification set modification process from making the same decisions (whether good or bad) twice for the same requirement qualification set. That is, a modification proposal that has been selected earlier for the same requirement qualification set should not be selected again (because it would lead to exactly the same results as produced earlier), and is marked rejected by the RQSM history maintenance process.

Figure 7.3 shows the types of information that an RQSM history maintenance process uses as input and the types of information that it produces as output.

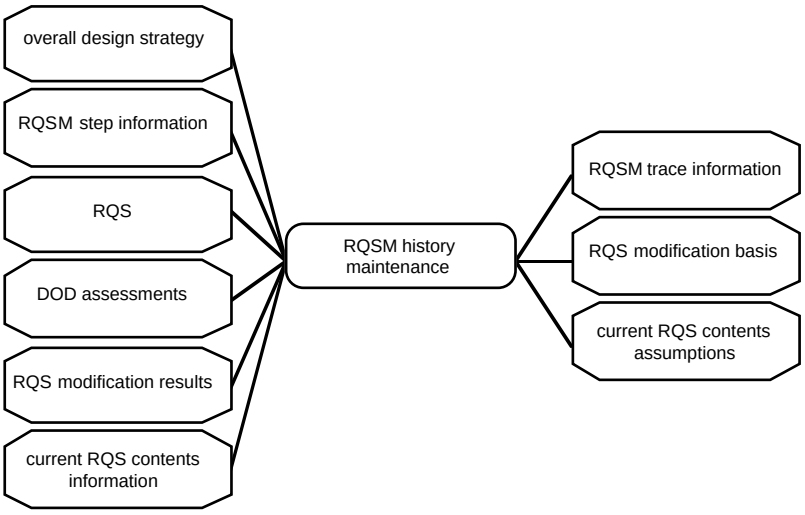


FIGURE 7.3. Input and output of an RQSM history maintenance process.

The following example, extending Example 7.2, is used to explain the types of input information used by an RQSM history maintenance process.

Example 7.3.

"The requirements included in the given set named *Definition Study Doc v0.1* are, among others, that the bicycle must be suited for riding in mountains and safe to be used by young children, and it must be low-priced. If the latter two requirements are irreconcilable, safety is preferred over price; the first requirement must be satisfied.

For the current set of design requirements, it is known that it includes the requirement that the bicycle must cost less than 400 euro. Of current interest for deductive refinement are the structural integrity requirements, such as the weight the bicycle must be able to carry and the forces it must be able to endure.

The requirements included in the set named *Definition Study Doc v0.4* are satisfiable. The design named *Design Doc v0.2* does not satisfy any of the requirements included in the set named *Definition Study Doc v0.3*.

The overall design strategy is to use earlier bicycle design cases as a basis for determining a set of design requirements. The next step in the manipulation process is to deductively refine the current set of design requirements."

All of the input information types of an RQSM history maintenance process have already been explained, except for current RQS contents information. This information type models epistemic information about the design requirement information included in the current requirement qualification set. (Here, it is only relevant to know which information is included,

and not which information is not included or not known to be included.) See Table 7.5 for the current RQS contents information in Example 7.3.

TABLE 7.5. Current RQS contents information in Example 7.3.

Kind	*Content*
Current RQS contents information	The current requirement qualification set is known to include a design requirement stating that the bicycle must cost less than 400 euro.

The following example, extending Example 7.3, is used to explain the types of output information produced by an RQSM history maintenance process.

Example 7.4.
"The current set is named *Definition Study Doc v0.5.* The current set is assumed to include the design requirement information that the bicycle cost less than 400 euro, that it must able to carry a load of 150 kg, and that its frame remains intact when it hits a wall with a speed of 50 km per hour. The requirements included in the set named *Definition Study Doc v0.4* are satisfiable. The design named *Design Doc v0.2* does not fulfil the set named *Definition Study Doc v0.3*.

In the design process, eight requirement qualification set manipulation states have been generated so far. The decision in the first, third, and fifth state was to deductively refine the current set. The decision in the second, fourth and sixth state was to modify the current set. The decision in the seventh state was to replace the current requirement qualification set (at that time, *Definition Study Doc v0.6*) by an earlier generated requirement qualification set (*Definition Study Doc v0.5*). The current overall design strategy is to use earlier bicycle design cases as a basis."

All of the output information types of an RQSM history maintenance process have already been explained, except for current RQS contents assumptions. This information type models meta-level assumptions about the design requirement information to be added to, or deleted from, the current requirement qualification set. See Table 7.6 for the current RQS contents assumptions in Example 7.4.

TABLE 7.6. Current RQS contents assumptions in Example 7.4.

Kind	*Content*
Current RQS contents assumptions	The current set is assumed to include the design requirement information that the bicycle must cost less than 400 euro, that it must able to carry a load of 150 kg, and that its frame remains intact when it hits a wall with a speed of 50 km per hour.

Current requirement qualification set maintenance
A current requirement qualification set maintenance process aims to record the design requirement information included in the current requirement qualification set and to provide up-to-date design requirement information. The design requirement information included in the current requirement qualification set may change due to the replacement of the current requirement qualification set by another set, the application of modifications to the current set, and the deductive refinement of the current set.

Figure 7.4 shows the type of information that a current requirement qualification set maintenance process uses as input and that it produces as output. This type has already been introduced in Chapter 6.

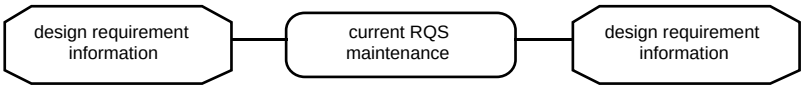


FIGURE 7.4. Input and output of a current requirement qualification set maintenance process.

Deductive requirement qualification set refinement
A deductive requirement qualification set refinement process deductively refines the current requirement qualification set on the basis of a domain-specific theory. Deductive refinement makes design requirements and relations between design requirements explicit that follow from the design requirement information already available and the domain-specific theory.

Figure 7.5 shows the type of information that a deductive requirement qualification set refinement process uses as input and that it produces as output. This type has already been introduced in Chapter 6.

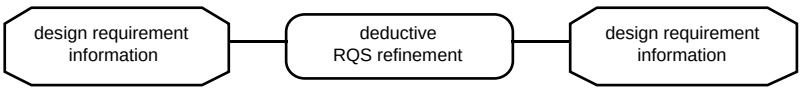


FIGURE 7.5. Input and output of a deductive requirement qualification set refinement process.

7.1.1.2 Process abstraction levels

A requirement qualification set manipulation process has been described to be composed of four processes: (1) a requirement qualification set modification process to determine appropriate modifications to the current requirement qualification set, (2) an RQSM history maintenance process to maintain historical information about requirement qualification set manipulation, (3) a current requirement qualification set maintenance process to maintain the design requirement information included in the current requirement qualification set, and (4) a deductive requirement qualification set refinement process to deductively refine the design requirement information included in the current requirement qualification set.

This composition is modelled in GDM by four abstraction relations; Figure 7.6 shows these relations between the component RQSM and the respective components RQS modification, RQSM history maintenance, current RQS maintenance and deductive RQS refinement.

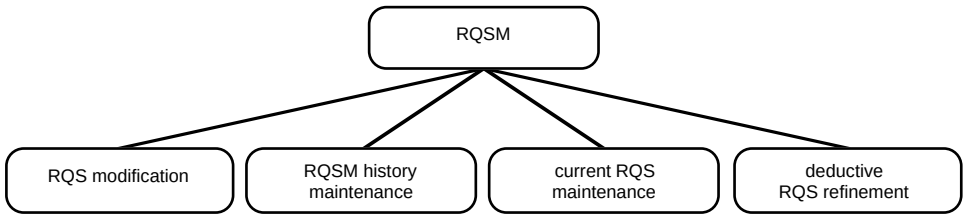


FIGURE 7.6. Two levels of abstraction for an RQS manipulation process.

7.1.2 Composition of processes

This section describes the way in which a requirement qualification set manipulation process is composed of lower-level processes, in terms of possibilities for exchange of information between processes, and in terms of task control knowledge used to control both the processes and the information exchange.

7.1.2.1 Information exchange

In a requirement qualification set manipulation process, all five processes involved (i.e., the process as a whole and its four sub-processes) exchange information. As explained in Chapter 5, a distinction is made between the information exchange between processes at different levels of process abstraction, and the information exchange between processes at the same level of process abstraction. Figure 7.7 gives a graphical overview of the exchange of information within a requirement qualification set manipulation process.

Firstly, an RQS manipulation process transfers the information it receives as input (the given overall design strategy, requirement qualification set and DOD assessments) to its RQS manipulation history maintenance sub-process (see Table 7.7). This information transfer enables explicit recording of the input information with which the RQS manipulation process has been activated (in a new manipulation state to be generated by the RQS manipulation history maintenance process).

TABLE 7.7. Information transfer from an RQS manipulation process to its sub-processes.

Source	*Destination*	*Information link*	*Information type*
RQSM	RQSM history maintenance	Overall design strategy for RQSM	Overall design strategy
RQSM	RQSM history maintenance	RQS for RQSM	RQS
RQSM	RQSM history maintenance	DOD assessments for RQSM	DOD assessments

Secondly, the four sub-processes of RQS manipulation exchange information with each other (see Table 7.8). RQSM trace information is transferred from RQSM history maintenance to RQS modification, as is the current RQS modification basis. If the current requirement qualification set has to be replaced by another set, current RQS contents assumptions are transferred from RQSM history maintenance to current RQS maintenance.

Information about the next RQSM step to be taken is transferred from RQS modification to RQSM history maintenance, as are the current RQS modification results. A new focus for deductive refinement of the current requirement qualification set is transferred from RQS modification to deductive RQS refinement.

When it is to be stored, the current RQS contents information is transferred from current RQS maintenance to RQSM history maintenance. When it is to be refined, the current design requirement information is transferred from current RQS maintenance to deductive RQS refinement. Finally, results of refining the current requirement qualification set are transferred from deductive RQS refinement to current RQS maintenance.

TABLE 7.8. Information exchange between sub-processes of an RQS manipulation process.

Source	*Destination*	*Information link*	*Information type*
RQSM history maintenance	RQS modification	Current RQSM trace information	RQSM trace information
RQSM history maintenance	RQS modification	Current RQS modification basis	RQS modification basis
RQSM history maintenance	Current RQS maintenance	Current RQS contents assumptions to be used	Current RQS assumptions
RQS modification	RQSM history maintenance	Current RQSM step information	RQSM step information
RQS modification	RQSM history maintenance	Current RQS modification results	RQS modification results
RQS modification	Deductive RQS refinement	Current deductive RQS refinement focus	Deductive RQS refinement focus
Current RQS maintenance	RQSM history maintenance	Current RQS contents information to be used	Current RQS contents information
Current RQS maintenance	Deductive RQS refinement	Current design requirement information	Design requirement information
Deductive RQS refinement	Current RQS maintenance	Refined design requirement information	Design requirement information

Thirdly, part of the information that the four sub-processes of RQS manipulation produced as output is transferred to become output of the RQS manipulation process (see Table 7.9). RQSM process evaluations are transferred from RQS modification, and appropriate requirement qualification sets and their assessments from RQSM history maintenance.

TABLE 7.9. Information transfer to an RQS manipulation process from its sub-processes.

Source	*Destination*	*Information link*	*Information type*
RQS modification	RQSM	Process evaluations from RQSM	RQSM process evaluations
RQSM history maintenance	RQSM	RQS from RQSM	RQS
RQSM history maintenance	RQSM	RQS assessments from RQSM	RQS assessments

The possibilities for information exchange between processes involved in RQS manipulation, as shown in Tables 7.7 to 7.9, are modelled in GDM by information links. Figure 7.7 shows a pictorial representation of the information links within the component RQSM.

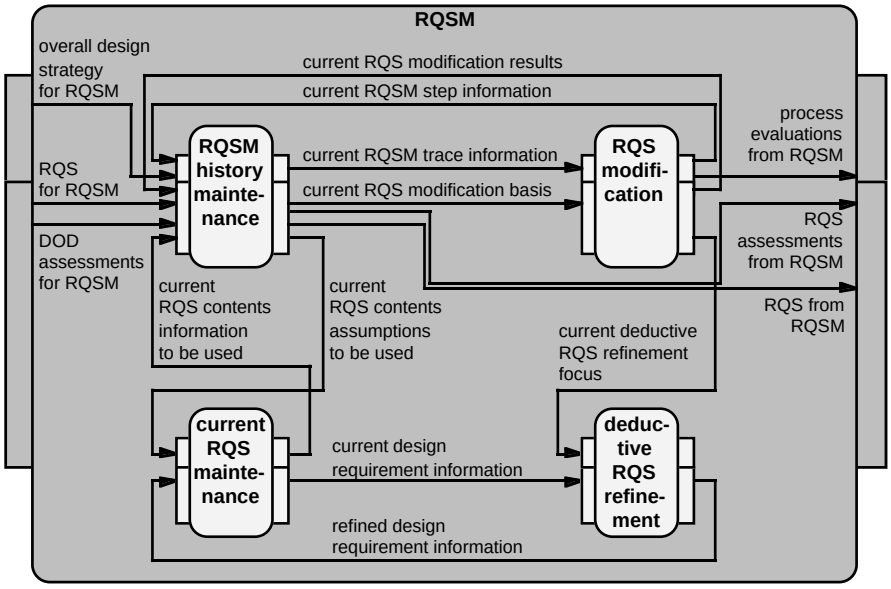


FIGURE 7.7. Information exchange between processes involved in RQS manipulation.

7.1.2.2 Task control

There are many ways in which a requirement qualification manipulation process can be controlled. The generic control method described in this section can be used effectively in any design application, but it is not necessarily the most efficient. Note that in specific situations, other (more specific) control methods may be more suitable.

Start of a requirement qualification set manipulation process

When a requirement qualification set manipulation process starts, most likely receiving new input information, the following actions are taken:

- the given overall design strategy, requirement qualification set (if any) and design object description assessments are transferred from the input of the RQS manipulation process to the input of RQSM history maintenance,
- the task control focus of RQSM history maintenance is set to be the commencement of the RQS manipulation process,
- RQSM history maintenance is activated to store the received information.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if the component RQSM is in its starting state, then in its next state the component RQSM history maintenance will have been activated with task control focus commencement, and the mediating links overall design strategy for RQSM, RQS for RQSM and DOD assessments for RQSM will have been updated.

Termination of RQSM history maintenance

Which actions are to be taken when RQSM history maintenance has terminated its activity depends on its task control focus. If RQSM history maintenance has terminated and its task control focus is to commence the RQS manipulation process or to process deductive refinements of the current requirement qualification set, then the following actions are taken:

- the current RQSM trace information and the current RQS modification basis are transferred from the output of RQSM history maintenance to the input of RQS modification,
- RQS modification is activated to determine the next RQS manipulation step.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component RQSM history maintenance is idle after having been active with task control focus commencement (denoting the start of the RQS manipulation process) or processing (denoting the act of processing deductive refinements of the current requirement qualification set), then in its next state the component RQS modification will have been activated, and the private links current RQSM trace information and current RQS modification basis will have been updated.

If RQSM history maintenance has terminated and its task control focus is to replace the current requirement qualification set with another set or to modify the current requirement qualification set, then the following actions are taken:

- assumptions about the design requirement information included in the current requirement qualification set are transferred from the output of RQSM history maintenance to the input of current RQS maintenance,
- current RQS maintenance is activated to update its design requirement information.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component RQSM history maintenance is idle after having been active with task control focus replacement (denoting the act of retrieving a requirement qualification set to replace the current requirement qualification set) or modification (denoting the act of modifying the current requirement qualification set), then in its next state the component current RQS maintenance will have been activated and the private link current RQS contents assumptions to be used will have been updated.

If RQSM history maintenance has terminated and its task control focus is termination of the RQS manipulation process, then the following actions are taken:

- the current evaluations about the RQS manipulation process are transferred from the output of RQS modification to the output of the RQS manipulation process,
- the results of the RQS manipulation process (requirement qualification sets and their assessments) are transferred from the output of RQSM history maintenance to the output of the RQS manipulation process,
- the RQS manipulation process is stopped.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component RQSM history maintenance is idle after having been active with task control focus termination (denoting the termination of the RQS manipulation process), then in its next state the mediating links process evaluations from RQSM, RQS from RQSM and RQS assessments from RQSM will have been updated and the component RQSM will have stopped.

Termination of RQS modification

Which actions are to be taken when RQS modification has terminated its activity depends on which evaluation criterion has succeeded. The possible evaluation criteria refer to the following RQS manipulation events: (1) termination of the RQS manipulation process, (2) inspection of the RQS manipulation history, (3) replacement of the current requirement qualification set by another set, (4) modification of the current requirement qualification set, and (5) deductive refinement of the current requirement qualification set.

If RQS modification has terminated and its evaluation criterion to terminate the RQS manipulation process has succeeded, then the following actions are taken:

- the current RQSM step information and the current results of RQS modification are transferred from the output of RQS modification to the input of RQSM history maintenance,
- the task control focus of RQSM history maintenance is set to *termination*,
- RQSM history maintenance is activated to store the current RQS modification results and RQSM step information.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component RQS modification is idle after having been active, and its evaluation criterion termination succeeds (denoting the termination of the RQS manipulation process), then in its next state the task control focus of the component RQSM history maintenance will have been set to termination, the component RQSM history maintenance will have been activated and the private links current RQSM step information and current RQS modification results will have been updated.

If RQS modification has terminated and its evaluation criterion to query and retrieve the RQS manipulation history has succeeded, then the following actions are taken:

- the current RQSM step information and the current results of RQS modification (including queries for historical information) are transferred from the output of RQS modification to the input of RQSM history maintenance,
- the task control focus of RQSM history maintenance is set to *query and retrieval*,
- RQSM history maintenance is activated to store the current RQS modification results and RQSM step information.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component RQS modification is idle after having been active, and its evaluation criterion query-and-retrieval succeeds (denoting that the RQS manipulation history is to be inspected), then in its next state the task control focus of the component RQSM history maintenance will have been set to query-and-retrieval, the component RQSM history maintenance will have been activated and the private links current RQSM step information and current RQS modification results will have been updated.

If RQS modification has terminated and its evaluation criterion to replace the current requirement qualification set has succeeded, then the following actions are taken:

- the current RQSM step information and the current results of RQS modification are transferred from the output of RQS modification to the input of RQSM history maintenance,
- the task control focus of RQSM history maintenance is set to *replacement*,
- RQSM history maintenance is activated to store the current RQS modification results and RQSM step information.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component RQS modification is idle after having been active, and its evaluation criterion replacement succeeds (denoting that the current requirement qualification set is to be replaced), then in its next state the task control focus of the component RQSM history maintenance will have been set to replacement, the component RQSM history maintenance will have been activated and the

private links current RQSM step information and current RQS modification results will have been updated.

If RQS modification has terminated and its evaluation criterion to modify the current requirement qualification set has succeeded, then the following actions are taken:

- the current RQSM step information and the current results of RQS modification are transferred from the output of RQS modification to the input of RQSM history maintenance,
- the task control focus of RQSM history maintenance is set to *modification*,
- RQSM history maintenance is activated to store the current RQS modification results and RQSM step information.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component RQS modification is idle after having been active, and its evaluation criterion modification succeeds (denoting that the current requirement qualification set is to be modified), then in its next state the task control focus of the component RQSM history maintenance will have been set to modification, the component RQSM history maintenance will have been activated, and the private links current RQSM step information and current RQS modification results will have been updated.

If RQS modification has terminated and its evaluation criterion to deductively refine the current requirement qualification set has succeeded, then the following actions are taken:

- the current RQSM step information and the current results of RQS modification are transferred from the output of RQS modification to the input of RQSM history maintenance,
- the current deductive RQS refinement focus is transferred from the output of RQS modification to the input of deductive RQS refinement,
- the current design requirement information is transferred from the output of current RQS maintenance to the input of deductive RQS refinement,
- deductive RQS refinement is activated to refine the current requirement qualification set on the basis of the given refinement focus.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component RQS modification is idle after having been active, and its evaluation criterion deductive-refinement succeeds (denoting that the current requirement qualification set is to be deductively refined), then in its next state the component deductive RQS refinement will have been activated, and the private links current RQSM step information, current RQS modification results, current design requirement information and current deductive RQS refinement focus will have been updated.

Termination of current RQS maintenance
Which actions are to be taken when current RQS maintenance has terminated its activity depends on which evaluation criterion of RQS modification has succeeded. The evaluation criteria to be considered are: (1) replacement of the current requirement qualification set, (2) modification of the current requirement qualification set, and (3) deductive refinement of the current requirement qualification set.

If current RQS maintenance has terminated and the evaluation criterion of RQS modification to replace or to modify the current requirement qualification set has succeeded, then the following actions are taken:

- the current RQSM trace information and the current RQS modification basis are transferred from the output of RQSM history maintenance to the input of RQS modification,
- RQS modification is activated to determine the next RQS manipulation step.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component current RQS maintenance is idle after having been active and the evaluation criterion replacement (denoting the act of replacing the current requirement qualification set) or the evaluation criterion modification (denoting the act of modifying the current requirement qualification set) of the component RQS modification has succeeded, then in its next state the component RQS modification will have been activated, and the private links current RQSM trace information and current RQS modification basis will have been updated.

If current RQS maintenance has terminated and the evaluation criterion of RQS modification to deductively refine the current requirement qualification set has succeeded, then the following actions are taken:

- the epistemic information about the contents of the deductively refined current requirement qualification set is transferred from the output of current RQS maintenance to the input of RQSM history maintenance,
- the task control focus of RQSM history maintenance is set to *processing*,
- RQSM history maintenance is activated to store the epistemic information about the contents of the current requirement qualification set.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component current RQS maintenance is idle after having been active and the evaluation criterion deductive-refinement (denoting the act of deductively refining the current requirement qualification set) of the component RQS modification has succeeded, then in its next state the component RQSM history maintenance will have been activated and the private link current RQS contents information to be used will have been updated.

Termination of deductive RQS refinement
If deductive RQS refinement has terminated, then the following actions are taken:

- the design requirement information included in the deductively refined current requirement qualification set is transferred from the output of deductive RQS refinement to the input of current RQS maintenance,
- current RQS maintenance is activated to store the given design requirement information.

This control knowledge is modelled in GDM by task control knowledge within the component RQSM, declaring that if, in the current state of the component RQSM, the component deductive RQS refinement is idle after having been active, then in its next state the component current RQS maintenance will have been activated, and the private link refined design requirement information will have been updated.

7.2 Knowledge Composition

This section describes knowledge structures identified in requirement qualification set manipulation at different levels of abstraction, and the composition of these knowledge structures.

7.2.1 Reflection levels

In a requirement qualification set manipulation process, the following three reflection levels are distinguished:

- The *first meta-level* includes information about design requirements. This level also includes deductive knowledge to derive implicit design requirement information, such as relations between design requirements.

- The *second meta-level*, directly above the first meta-level, includes information about requirement qualification sets and about the relations between specific requirement qualification sets and design object descriptions. This level also includes knowledge to assess requirement qualification sets, as well as strategic knowledge to determine possible modifications of requirement qualification sets.

- The *third meta-level*, directly above the second meta-level, includes information about overall design strategies and about the trace and process evaluations of a requirement qualification set manipulation process. This level also includes knowledge to determine strategies for the manipulation of requirement qualification sets.

7.2.2 Knowledge structures and composition

This section describes the composition and structure of the generic knowledge related to requirement qualification set manipulation, and it shows how they are modelled in GDM. For the sake of comprehension, the boxes with thick lines in the figures indicate the application specific knowledge structures of GDM that are candidates for refinement when using GDM to model a requirement qualification set manipulation process in a specific application domain.

The presented examples extend the bicycle design example. In these examples, terms and expressions shown in italics are part of the model of the application domain, not of GDM.

7.2.2.1 Structure and composition of knowledge at the first meta-level

The first meta-level includes *design requirement information*, describing requirements, qualified requirements, and their relations. Refer to Chapter 6 for the composition and structure of the information type design requirement information.

7.2.2.2 Structure and composition of knowledge at the second meta-level

The second meta-level includes the following main types of information: (1) epistemic information about the contents of the current requirement qualification set, (2) information used as a basis for modification of the current requirement qualification set, (3) assumptions about the contents of the current requirement qualification set, and (4) information resulting from the process of modifying the current requirement qualification set.

Current RQS contents information. Figure 7.8 shows the composition of the information type current RQS contents information, which models epistemic information about the contents of the current requirement qualification set. This information type is used in situations where the contents of the current requirement qualification set have changed due to modification or deductive refinement; in these situations, the new contents have to be reflected upwards, included in a new requirement qualification set, and stored as part of the manipulation history.

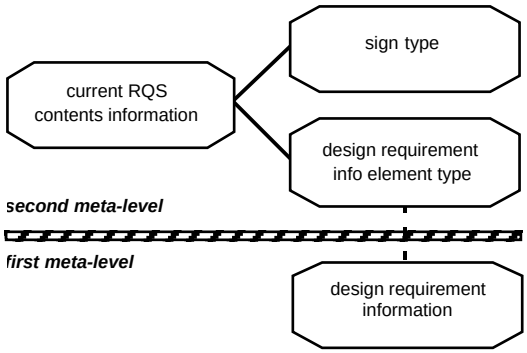


FIGURE 7.8. Composition of information type current RQS contents information.

Figure 7.9 shows the structure of the information type current RQS contents information. It contains the relation is-currently-included, which has two arguments of the sorts design-requirement-info-element and sign, respectively. An atom is-currently-included(*design-requirement-info-element1*, *sign1*) specifies that the design requirement information element *design-requirement-info-element1*, with sign *sign1* as its truth value, is included in the current requirement qualification set.

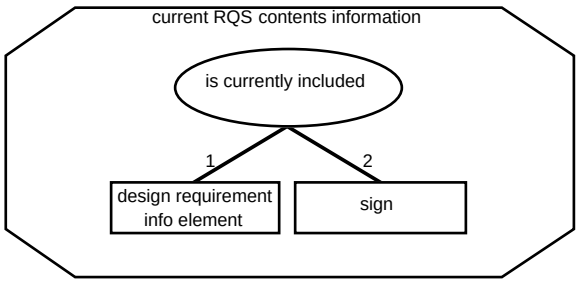


FIGURE 7.9. Structure of information type current RQS contents information.

Example 7.5.
"The design requirement information included in the current set is that the bicycle must be suited for riding in mountains (requirement R1), low-priced (requirement R2) and safe to be used by young children (requirement R3)."

is-currently-included(is-defined-as(*R1*, *is-suitable-for-terrain-type*(*bicycle1*, *mountains*)), pos)
is-currently-included(is-defined-as(*R2*, *is-price-level*(*bicycle1*, *low*)), pos)
is-currently-included(is-defined-as(*R3*, *is-safe-to-be-used-by*(*bicycle1*, *young-children*)), pos)

RQS modification basis. Figure 7.10 shows the composition of the information type RQS modification basis, which models information that is used as a basis for modification of the current requirement qualification set. This information, together with RQSM trace information (explained in the next sub-section), forms the input of the component RQS modification.

Refer to Chapter 6 for the composition and structure of the information types design requirement information, design requirement info element type, RQS name type, RQS, RQS solution information, RQS assessments and DOD assessments. In the following, the other information types are explained to which the information type RQS modification basis refers.

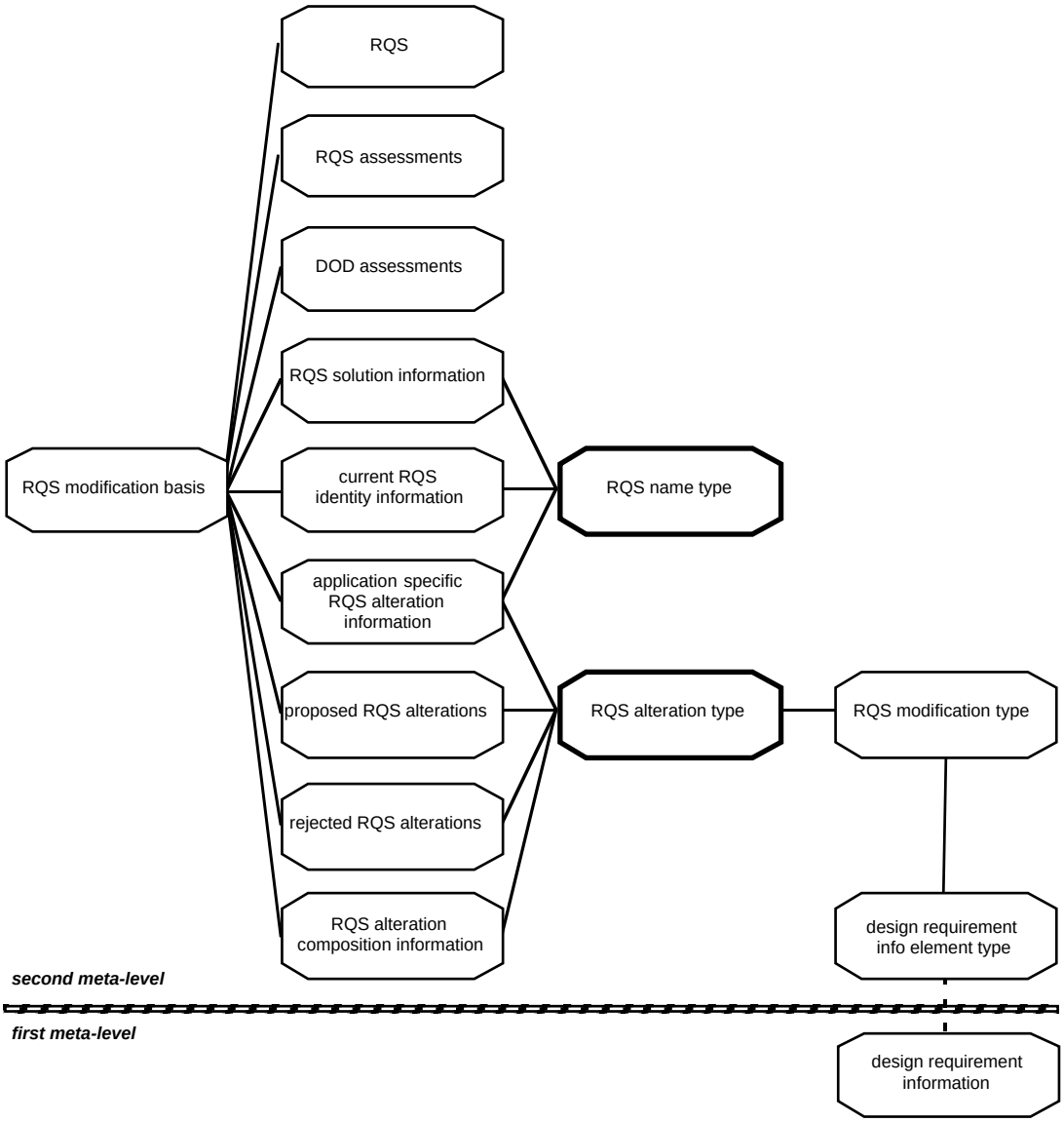


FIGURE 7.10. Composition of information type RQS modification basis.

Figure 7.11 shows the structure of the information type current RQS identity information, which models information about the identity of the current requirement qualification set. It contains the relation is-current-RQS, which has one argument of the sort RQS-name. An atom is-current-RQS(*RQS-name1*) specifies that the requirement qualification set named *RQS-name1* is the current requirement qualification set (and subject to modification by the requirement qualification set modification process).

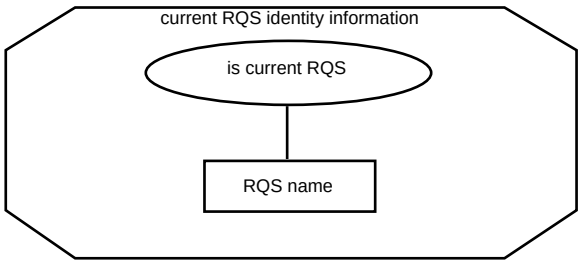


FIGURE 7.11. Structure of information type current RQS identity information.

Example 7.6.
"The current set is named *Definition Study Doc v0.5*."

is-current-RQS("*Definition Study Doc v0.5*")

An *RQS modification* specifies either the addition of design requirement information to the current requirement qualification set, or the deletion of design requirement information from the current requirement qualification set. Objects of the sort RQS-modification within the information type RQS modification type are used to model RQS modifications.

This information type also specifies two functions, addition-of and deletion-of, which both have one argument of the sort design-requirement-info-element and the sort RQS-modification as their range. The function addition-of is used to model the addition of specific design requirement information to the current requirement qualification set, whereas the function deletion-of is used to model the deletion of specific design requirement information from the current requirement qualification set.

An *RQS alteration* specifies a set of one or more modifications to be made to the current requirement qualification set. Objects of the sort RQS-alteration within the information type RQS alteration type are used to model RQS alterations. An alteration may be formulated as a single RQS modification (modelled by an object of the sub-sort RQS-modification).

The information type application specific RQS alteration information models application specific information about specific requirement qualification set alterations. For example, a relation specified within this information type can be used to model preferences for different alterations.

Figure 7.12 shows the structure of the information type RQS alteration composition information, which models information about the modifications included in a specific alteration to the current requirement qualification set. It contains the relation includes-RQS-modification, which has two arguments of the sorts RQS-alteration and RQS-modification, respectively. An atom includes-RQS-modification(*RQS-alteration1*, *RQS-modification1*) specifies that the alteration *RQS-alteration1* includes the modification *RQS-modification1*.

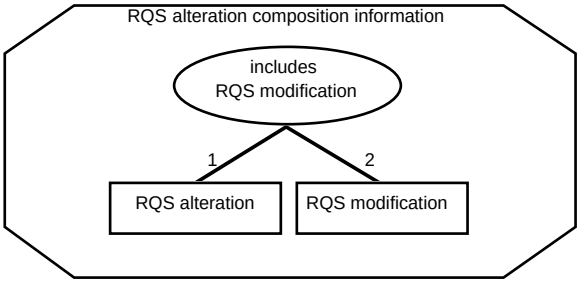


FIGURE 7.12. Structure of information type RQS alteration composition information.

Example 7.7.
"One possibility (option 1) to alter the current set of design requirements is to replace the requirement (R2) that the bicycle be low priced by a requirement (R4) that the bicycle's price must be less than 400 euro."

includes-RQS-modification(*AlterationOption1*,
  deletion-of(is-defined-as(*R2*, *is-price-level*(*bicycle1*, *low*))))
includes-RQS-modification(*AlterationOption1*,
  addition-of(is-defined-as(*R4*, *price-of*(*bicycle1*, *euro*) < 400)))

Figure 7.13 shows the structure of the information type proposed RQS alterations, which models information about proposed alterations to the current requirement qualification set. It contains the relation is-proposed-RQS-alteration, which has one argument of the sort RQS-alteration. An atom is-proposed-RQS-alteration(*RQS-alteration1*) specifies that the alteration *RQS-alteration1* is an existing proposal to modify the current requirement qualification set.

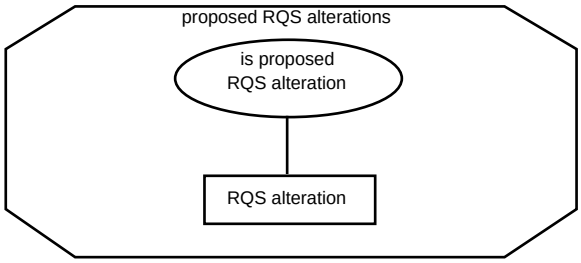


FIGURE 7.13. Structure of information type proposed RQS alterations.

Example 7.8.
"A current proposal (option 1) is to replace the requirement (R2) that the bicycle be low priced by a requirement (R4) that the bicycle's price must be less than 400 euro."

is-proposed-RQS-alteration(*AlterationOption1*)

Figure 7.14 shows the structure of the information type rejected RQS alterations, which models information about rejected alterations to the current requirement qualification set. (These alterations have been applied earlier to the same requirement qualification set, so to prevent them from being applied again, they have been marked as rejected.) The information type contains the relation is-rejected-RQS-alteration, which has one argument of the sort RQS-alteration. An atom is-rejected-RQS-alteration(*RQS-alteration1*) specifies that the alteration *RQS-alteration1* is rejected as a proposal to modify the current requirement qualification set.

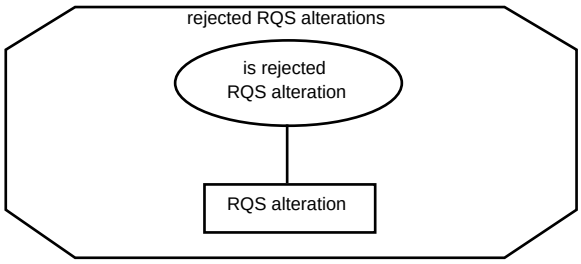


FIGURE 7.14. Structure of information type rejected RQS alterations.

Example 7.9.
"A proposal has been rejected to remove the requirement (R3) that the bicycle must be safe to be used by young children."

is-rejected-RQS-alteration(
  deletion-of(is-defined-as(*R3*, *is-safe-to-be-used-by*(*bicycle1*, *young-children*))))

Current RQS contents assumptions. Figure 7.15 shows the composition of the information type current RQS contents assumptions, which models assumptions about the contents of the current requirement qualification set.

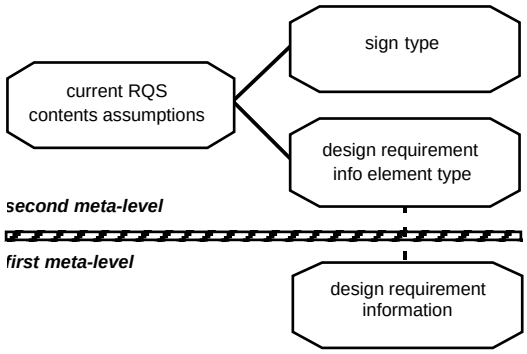


FIGURE 7.15. Composition of information type current RQS contents assumptions.

Figure 7.16 shows the structure of the information type current RQS contents assumptions. It contains two relations, is-to-be-asserted and is-to-be-retracted, which both have two arguments of the sorts design-requirement-info-element and sign, respectively. An atom is-to-be-asserted(*design-requirement-info-element1*, *sign1*) specifies that the design requirement information *design-requirement-info-element1*, with sign *sign1* as its truth value, is assumed to be included in the current requirement qualification set. An atom is-to-be-retracted(*design-requirement-info-element1*, *sign1*) specifies that the design requirement information *design-requirement-info-element1*, with sign *sign1* as its truth value, is assumed not to be included in the current requirement qualification set.

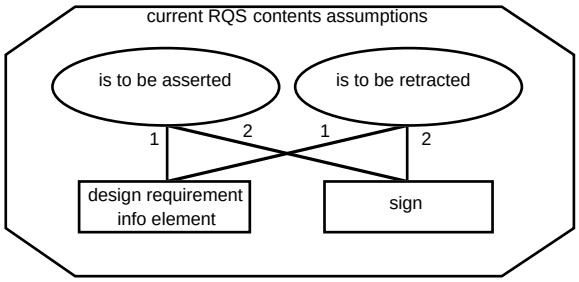


FIGURE 7.16. Structure of information type current RQS contents assumptions.

Example 7.10.
"The current set of design requirements is assumed to include the fact that the requirement that the bicycle is safe to be used by young children must be satisfied, and it is assumed not to include the fact that the requirement that the bicycle is high priced must be satisfied."

is-to-be-asserted(is-to-be-satisfied(*is-safe-to-be-used-by*(*bicycle1*, *young-children*)), pos)
is-to-be-retracted(is-to-be-satisfied(*is-price-level*(*bicycle1*, *high*)), pos)

RQS modification results. Figure 7.17 shows the composition of the information type RQS modification results, which models information resulting from the process of modifying the current requirement qualification set. This information, together with RQSM step information (explained in the next sub-section), forms the output of the component RQS modification.

The information types design requirement information, design requirement info element type, sign type, RQS name type, RQS assessments, RQS solution information, RQS modification type, RQS alteration type, RQS alteration composition information, proposed RQS alterations, and application specific RQS alteration information have already been explained. In the following, the other information types are explained to which the information type RQS modification results refers.

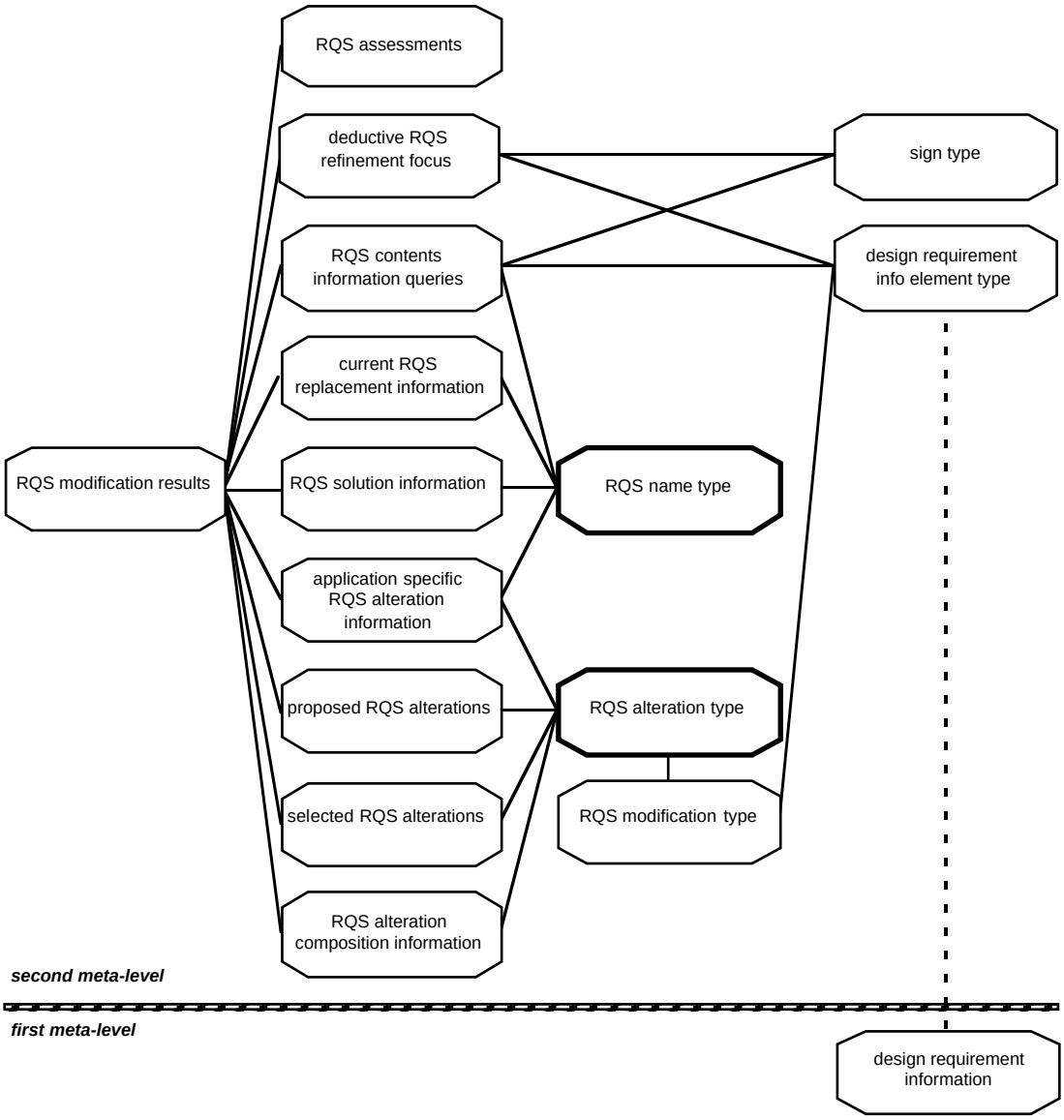


FIGURE 7.17. Composition of information type RQS modification results.

Figure 7.18 shows the structure of the information type current RQS replacement information, which models information about the identity of a requirement qualification set that is to replace the current requirement qualification set. It contains the relation is-replacement-for-current-RQS, which has one argument of the sort RQS-name. An atom is-replacement-for-current-RQS(*RQS-name1*) specifies that the requirement qualification set named *RQS-name1* is to replace the current requirement qualification set.

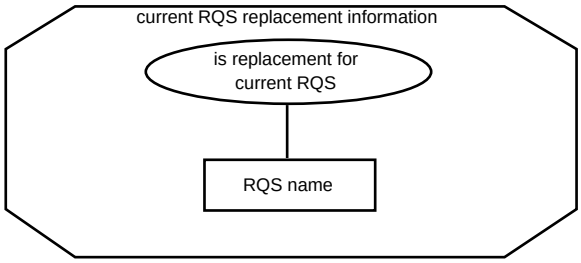


FIGURE 7.18. Structure of information type current RQS replacement information.

Example 7.11.
"The current set has to be replaced by the set of design requirements named *Definition Study Doc v0.5*."

is-replacement-for-current-RQS("*Definition Study Doc v0.5*")

Figure 7.19 shows the structure of the information type RQS contents information queries, which models queries for information about the contents of requirement qualification sets. It contains two relations: the relation includes-which-design-requirement-information has one argument of the sort RQS-name, and the relation is-included-in-which-RQSs has two arguments of the sorts design-requirement-info-element and sign, respectively.

An atom includes-which-design-requirement-information(*RQS-name1*) specifies a query for the design requirement information included in the requirement qualification set named *RQS-name1*. An atom is-included-in-which-RQSs(*design-requirement-info-element1*, *sign1*) specifies a query for the requirement qualification sets that include the design requirement information *design-requirement-info-element1* with sign *sign1* as its truth value.

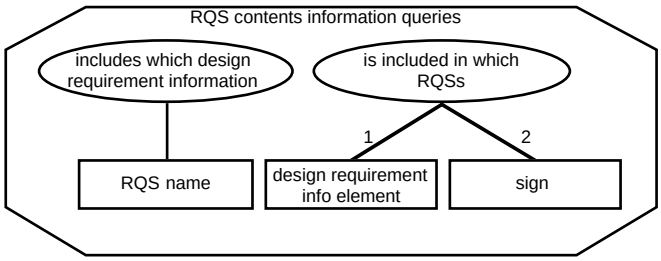


FIGURE 7.19. Structure of information type RQS contents information queries.

Example 7.12.
"There is currently a query for the design requirement information included in the set of design requirements named *Definition Study Doc v0.1*. There is also currently a

query for the design requirement sets that include the design requirement information that the bicycle must be able to carry a load of 150 kg."

includes-which-design-requirement-information("*Definition Study Doc v0.1*")
is-included-by-which-RQSs(
for-all(*N*, l-implies(*has-tolerance*(*bicycle1*, *load*, *weight*(*N*, *kg*)), *ge*(*N*, 150))),
pos)

Figure 7.20 shows the structure of the information type selected RQS alterations, which models information about selected alterations to the current requirement qualification set. It contains the relation is-selected-RQS-alteration, which has one argument of the sort RQS-alteration. An atom is-selected-RQS-alteration(*RQS-alteration1*) specifies that the alteration *RQS-alteration1* is selected (i.e., this alteration is to be applied in order to modify the current requirement qualification set).

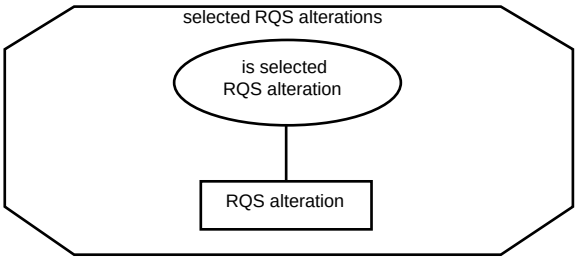


FIGURE 7.20. Structure of information type selected RQS alterations.

Example 7.13.
"The proposal is selected to replace the requirement (R2) that the bicycle be low priced by a requirement (R4) that the bicycle's price must be less than 400 euro."

is-selected-RQS-alteration(*AlterationOption1*)

Figure 7.21 shows the structure of the information type deductive RQS refinement focus, which models information about the current focus for the deductive refinement of the current requirement qualification set. It contains the relation is-part-of-deductive-RQS-refinement-focus, which has two arguments of the sorts design-requirement-info-element and sign, respectively. An atom is-part-of-deductive-RQS-refinement-focus(*design-requirement-info-element1*, *sign1*) specifies that the design requirement information element *design-requirement-info-element1* with sign *sign1* is part of the current focus for deductive refinement of the current requirement qualification set.

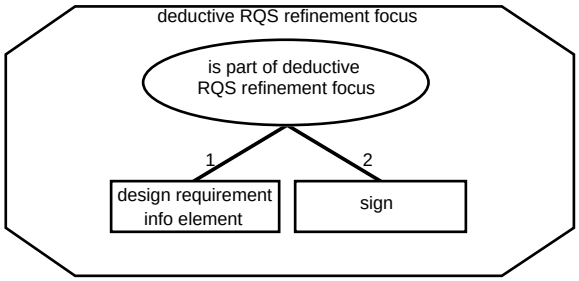


FIGURE 7.21. Structure of information type deductive RQS refinement focus.

Example 7.14.
"Of current interest for deductive refinement are the structural integrity requirements, such as the weight that the bicycle must be able to carry."

is-part-of-deductive-RQS-refinement-focus(
for-all(*N*, l-implies(*has-tolerance*(*bicycle1*, *load*, *weight*(*N*, *kg*)), *ge*(*N*, *MinVar*))), pos)

RQS modification process information. Figure 7.22 shows the composition of the information type RQS modification process information, which models information about the input and output of an RQS modification process. It refers to two information types, RQS modification basis and RQS modification results, which together model the basis for RQS modification and the results of RQS modification. (This information type is used to reflect RQS modification process information upward to the third meta-level.)

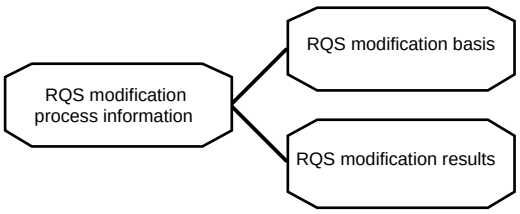


FIGURE 7.22. Composition of information type RQS modification process information.

7.2.2.3 Structure and composition of knowledge at the third meta-level

The third meta-level includes information about traces and steps of RQS manipulation processes.

RQSM trace information. Figure 7.23 shows the composition of the information type RQSM trace information, which models information about the trace of the requirement qualification set manipulation process.

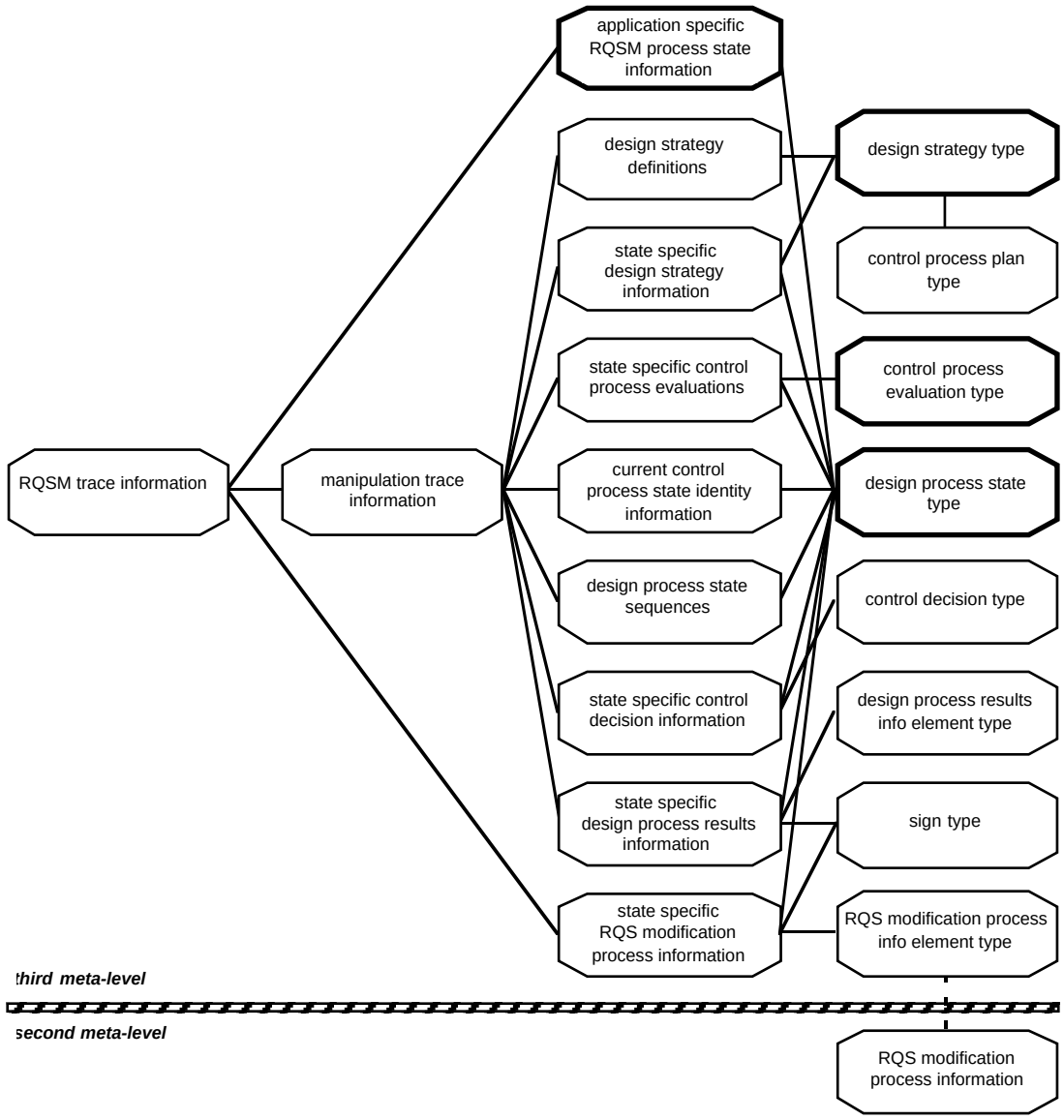


FIGURE 7.23. Composition of information type RQSM trace information.

The information types design process state type, design strategy name type, design strategy definitions, state specific design strategy information, design process results info element type, control process plan type, control process evaluation type, and sign type have been explained earlier. In the following, the other information types are explained to which the information type RQSM trace information refers.

The information type RQS modification process info element type models meta-descriptions of information about the process of modifying the current requirement qualification set. These meta-descriptions are obtained by the upward reflection of information about require-

ment qualification set modification (modelled by the information type RQS modification process information).

Figure 7.24 shows the structure of the information type state specific RQS modification process information, which models epistemic information about the requirement qualification set modification process. It contains the relation includes-RQS-modification-process-information, which has three arguments of the sorts design-process-state, RQS-modification-process-info-element and sign, respectively. An atom includes-RQS-modification-process-information(*design-process-state1*, *RQS-modification-process-info-element1*, *sign1*) specifies that the design process state *design-process-state1* includes the modification process information *RQS-modification-process-info-element1*, with the sign *sign1* as its truth value.

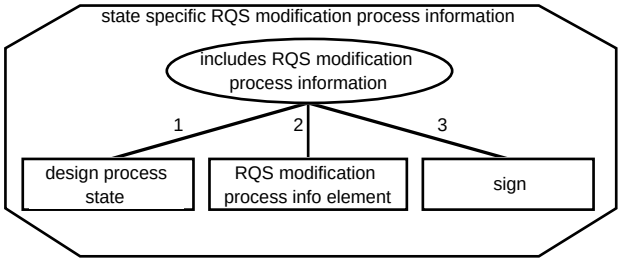


FIGURE 7.24. Structure of information type state specific RQS modification process information.

Example 7.15.
"The state of the requirement qualification set manipulation process designated State5 includes the information that there is a proposal to replace the requirement that the bicycle be low priced by one that the bicycle's price must be less than 400 euro."

includes-RQS-modification-process-information(*State5*,
  is-proposed-RQS-alteration(*AlterationOption1*), pos)
includes-RQS-modification-process-information(*State5*,
  involves-RQS-modification(*AlterationOption1*,
    deletion-of(is-defined-as(*R2*, *is-price-level*(*bicycle1*, *low*)))), pos)
includes-RQS-modification-process-information(*State5*,
  involves-RQS-modification(*AlterationOption1*,
    addition-of(is-defined-as(*R4*, *price-of*(*bicycle1*, *euro*) < 400))), pos)

Figure 7.25 shows the structure of the information type current control process state identity information, which models information about the identity of the current state of the control process. It contains the relation is-current-control-process-state, which has one argument of the sort design-process-state. An atom is-current-control-process-state(*design-process-state1*) specifies that the design process state *design-process-state1* is the current state of the control process.

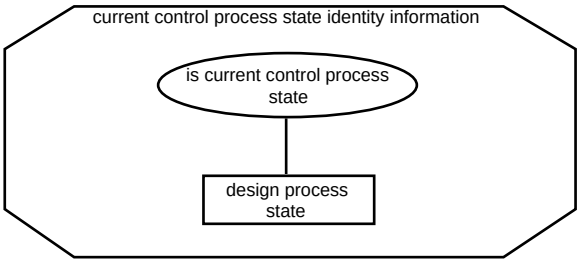


FIGURE 7.25. Structure of information type current control process state identity information.

Example 7.16.
"The current state of requirement qualification set manipulation is designated State8."

is-current-control-process-state(*State8*)

The information type control decision type models decisions about the steps taken in a requirement qualification set manipulation process or in a design object description manipulation process. Applied to a requirement qualification set manipulation process, the possible decisions are: replacement of the current requirement qualification set (by an earlier generated set), modification of the current requirement qualification set, deductive refinement of the current requirement qualification set, inspection of the requirement qualification set manipulation history, and termination of the requirement qualification set manipulation process. These different decisions are modelled by the objects replacement, modification, deductive-refinement, query-and-retrieval, and termination, respectively, all of the sort control-decision.

Figure 7.26 shows the structure of the information type state specific control decision information. It contains the relation includes-control-decision, which has two arguments of the sorts design-process-state and control-decision, respectively. An atom includes-control-decision(*design-process-state1*, *control-decision1*) specifies that the design process state *design-process-state1* includes the control decision *control-decision1* for the next step of the control process.

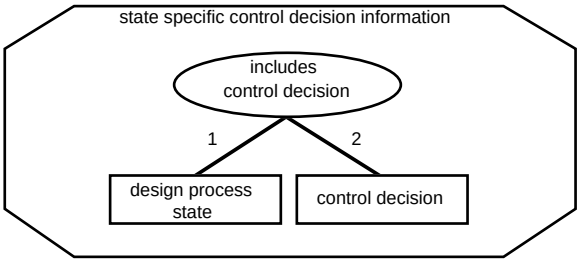


FIGURE 7.26. Structure of information type state specific control decision information.

Example 7.17.
"The decision taken in the state of the requirement qualification set manipulation process designated State1 was to deductively refine the current set of design requirements."

includes-control-decision(*State1*, deductive-refinement)

Figure 7.27 shows the structure of the information type state specific design process results information, which models information about the manipulation strategies in the different states of a design process. It contains the relation includes-design-process-results-information, which has three arguments of the sorts design-process-state, design-process-results-information, and sign, respectively. An atom includes-design-process-results-information(*design-process-state1*, *design-process-results-info-element1*, *sign1*) specifies that the design process state *design-process-state1* involves the design process results information *design-process-results-info-element1*, with the sign *sign1* as its truth value.

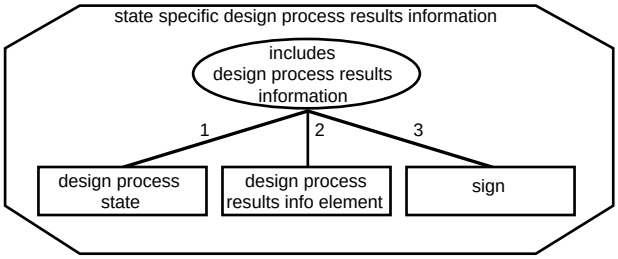


FIGURE 7.27. Structure of information type state specific design process results information.

Example 7.18.
"In the requirement qualification set manipulation process state designated State7, it has been established that determining the design requirements of the mountain bike has taken eight person hours so far."

includes-design-process-results-information(
State7, has-past-consumption-of(*person-hours*, 8))

Figure 7.28 shows the structure of the information type design process state sequences, which models information about the sequences of states of a design process. It contains the relation has-succeeding-design-process-state, which has two arguments that are both of the sort design-process-state. An atom has-succeeding-design-process-state(*design-process-state1*, *design-process-state2*) specifies that the design process state *design-process-state1* has the design process state *design-process-state2* as its successor.

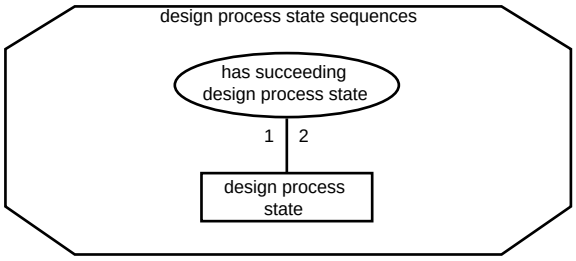


FIGURE 7.28. Structure of information type design process state sequences.

Example 7.19.
"The requirement qualification set manipulation process state designated State7 is followed by the state designated State8."

has-succeeding-design-process-state(*State7*, *State8*)

Figure 7.29 shows the structure of the information type state specific control process evaluations, which models control process evaluations of specific states of a control process. It contains one relation, includes-control-process-evaluation, which has two arguments of the sorts design-process-state and control-process-evaluation, respectively. An atom includes-control-process-evaluation(*design-process-state1*, *control-process-evaluation1*) specifies that the design process state *design-process-state1* includes *control-process-evaluation1* as an evaluation of the control process with respect to the overall design strategy.

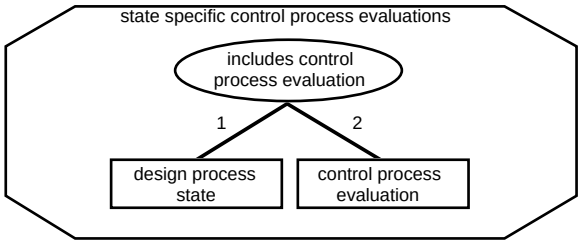


FIGURE 7.29. Structure of information type state specific control process evaluations.

Example 7.20.
"In the requirement qualification set manipulation process state designated State5, the overall design strategy has been evaluated to be incomplete."

includes-control-process-evaluation(*State5*, incomplete)

The information type application specific RQSM process state information models application specific information about individual states of a requirement qualification set manipulation

process and their relations. For example, a relation within this information type can be used to model the heuristics-based information that a specific process state is promising.

RQSM step information. Figure 7.30 shows the composition of the information type RQSM step information, which models information about the next step to be taken in a requirement qualification set manipulation process.

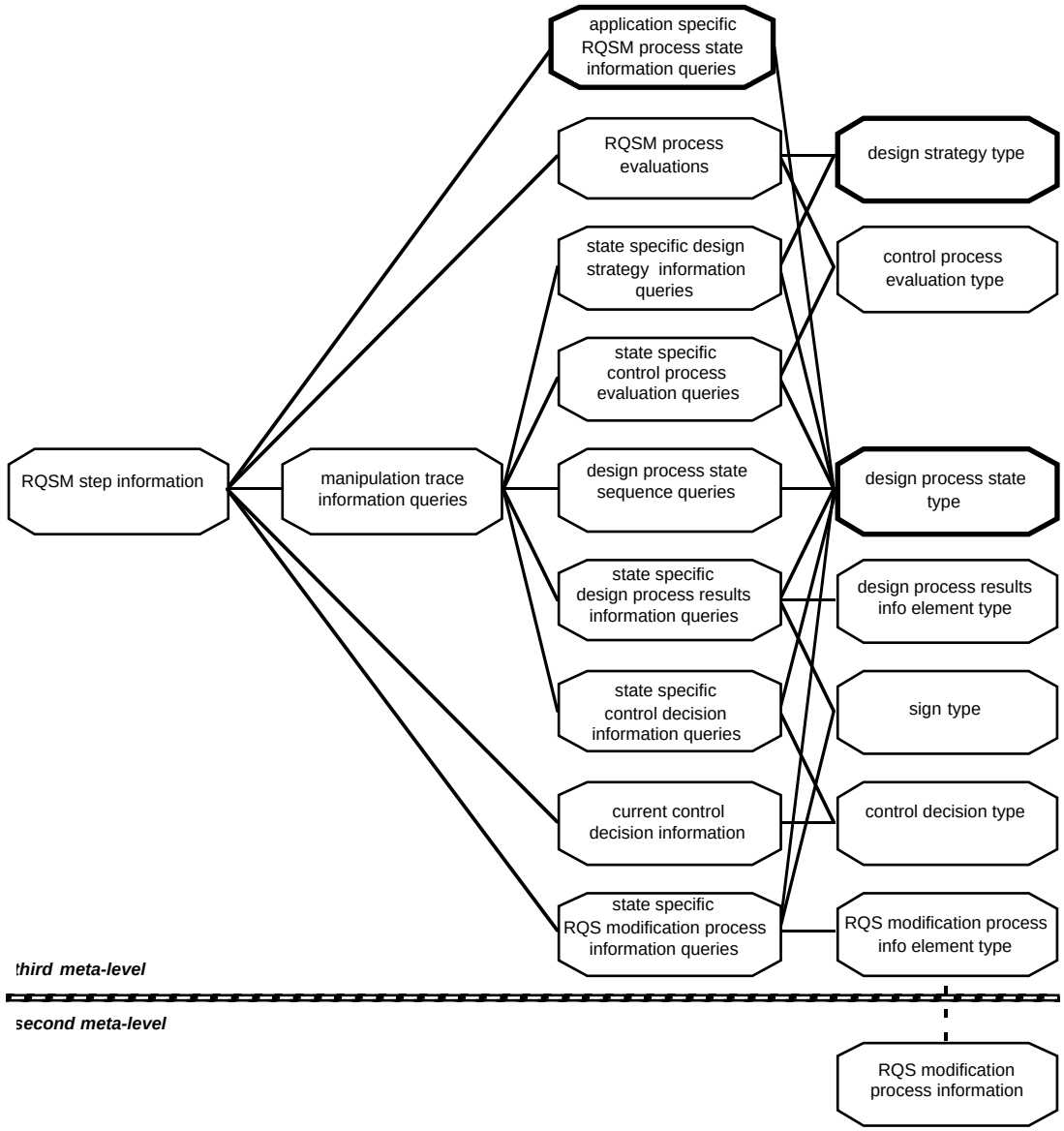


FIGURE 7.30. Composition of information type RQSM step information.

The information types RQS modification process information, sign type, RQS modification process info element type, design process state type, design strategy name type, control decision type, design process results info element type, and RQSM process evaluations have already been explained (the latter in Chapter 6). In the following, the other information types are explained to which the information type RQSM step information refers.

Figure 7.31 shows the structure of the information type current control decision information, which models information about the decision for the next step in the control process. It contains the relation is-current-control-decision, which has one argument of the sort control-decision. An atom is-current-control-decision(*control-decision1*) specifies that the decision for the next step in the manipulation process is *control-decision1*.

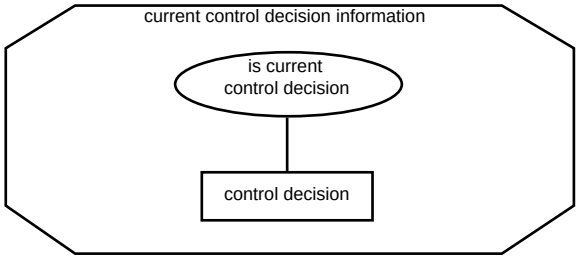


FIGURE 7.31. Structure of information type current control decision information.

Example 7.21.
"The decision for the next step of the requirement qualification set manipulation process is to modify the current set of design requirements."

is-current-control-decision(modification)

Figure 7.32 shows the structure of the information type state specific design strategy information queries, which models queries related to the design strategies involved in specific design process states. It contains two relations: the relation includes-which-design-strategy has one argument of the sort design-process-state, and the relation is-included-by-which-design-process-states has one argument of the sort design-strategy-name.

An atom includes-which-design-strategy(*design-process-state1*) specifies a query for the design strategy involved in the design process state *design-process-state1*. An atom is-included-by-which-design-process-states(*design-strategy-name1*) specifies a query for the design process states in which the design strategy *design-strategy-name1* is included.

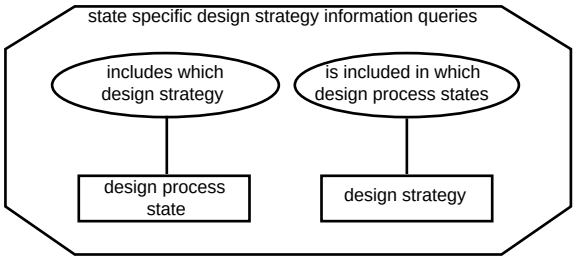


FIGURE 7.32. Structure of information type state specific design strategy information queries.

Example 7.22.
"There is currently a query for the design strategy that held at the time of the requirement qualification set manipulation process state designated State5. Furthermore, there is currently a query for the requirement qualification set manipulation process states in which the design strategy was to use earlier design cases."

has-which-design-strategy(*State5*)
is-strategy-in-which-design-process-states(*use-earlier-design-cases*)

Figure 7.33 shows the structure of the information type state specific control decision information queries, which models queries for the control decisions included in specific design process states. It contains two relations: the relation includes-which-control-decision has one argument of the sort design-process-state, and the relation is-decision-in-which-design-process-states has one argument of the sort control-decision.

An atom includes-which-control-decision(*design-process-state1*) specifies a query for the manipulation decision included in the design process state *design-process-state1*. An atom is-decision-in-which-design-process-states(*control-decision1*) specifies a query for the design process states which include the control decision *control-decision1*.

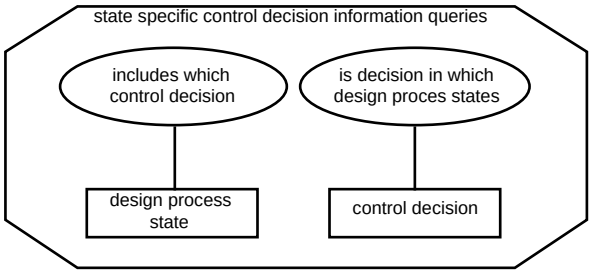


FIGURE 7.33. Structure of information type state specific control decision information queries.

Example 7.23.
"There is currently a query for the control decision included in the requirement qualification set manipulation process state named State1. Furthermore, there is a query for the requirement qualification set manipulation process states in which it has been decided to replace the current requirement qualification set."

includes-which-control-decision(*State1*)
is-decision-in-which-design-process-states(replacement)

Figure 7.34 shows the structure of the information type design process state sequence queries, which models queries for the sequence in which design process states have been generated. It contains two relations, is-preceded-by-which-design-process-state and is-succeeded-by-which-design-process-state, which both have one argument of the sort design-process-state.

An atom is-preceded-by-which-design-process-state(*design-process-state1*) specifies a query for the design process state that precedes the design process state *design-process-state1*. An atom is-succeeded-by-which-design-process-state(*design-process-state1*) specifies a query for the design process state that succeeds the design process state *design-process-state1*.

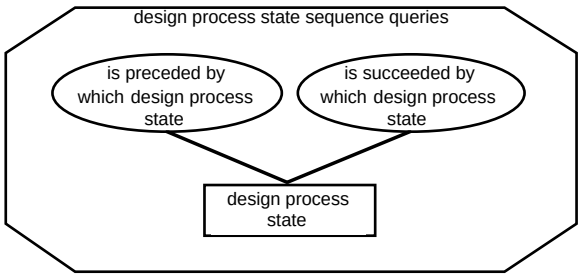


FIGURE 7.34. Structure of information type design process state sequence queries.

Example 7.24.
"There is currently a query for the requirement qualification set manipulation process state that succeeded the state designated State2."

is-succeeded-by-which-design-process-state(*State2*)

Figure 7.35 shows the structure of the information type state specific control process evaluation queries, which models queries related to the control process evaluations of specific design process states. It contains two relations: the relation includes-which-control-process-evaluation has one argument of the sort design-process-state, and the relation is-evaluation-of-which-design-process-states has one argument of the sort control-process-evaluation.

An atom includes-which-control-process-evaluation(*design-process-state1*) specifies a query for the evaluation of the control process that is included in the design process state *design-process-state1*. An atom is-evaluation-of-which-design-process-states(*control-process-evaluation1*) specifies a query for the design process states which include the control process evaluation *control-process-evaluation1*.

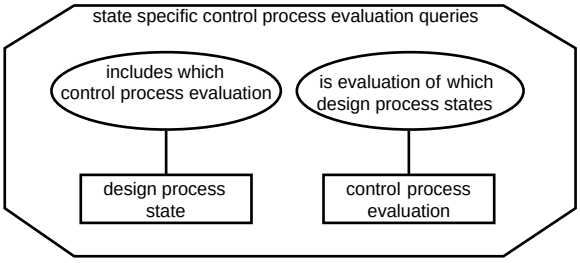


FIGURE 7.35. Structure of information type state specific control process evaluation queries.

Example 7.25.
"There is currently a query for the control process evaluation of the requirement qualification set manipulation process state designated State5. Furthermore, there is currently a query for the requirement qualification set manipulation process states which evaluated the overall design strategy to be completed with failure as a result."

includes-which-control-process-evaluation(*State5*)
is-evaluation-of-which-design-process-states(failed)

Figure 7.36 shows the structure of the information type state specific design process results information queries, which models queries for the design process results information included in a specific design process state. It contains two relations: the relation includes-which-design-process-results-information has one argument of the sort design-process-state, and the relation is-included-in-which-design-process-states has two arguments of the sorts design-process-results-info-element and sign, respectively.

An atom includes-which-design-process-results-information(*design-process-state1*) specifies a query for the design process results information included in the design process state *design-process-state1*. An atom is-included-in-which-design-process-states(*design-process-results-info-element1*, *sign1*) specifies a query for the design process states that include the design process results information element *design-process-results-info-element1* with sign *sign1* as its truth value.

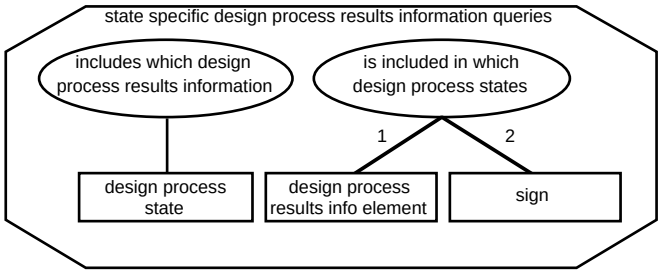


FIGURE 7.36. Structure of information type state specific design process results information queries.

Example 7.26.
"There is currently a query for the design process results information included in the requirement qualification set manipulation process state designated State6. Furthermore, there is currently a query for the designators of all past process states in which the requirement qualification set manipulation process was active."

includes-which-design-process-results-information(*State6*)
is-included-by-which-design-process-states(is-current-status(active), pos)

Figure 7.37 shows the structure of the information type state specific RQS modification process information queries, which models queries for the information about the requirement qualification set modification process information included in a specific design process state. It contains two relations: the relation includes-which-RQS-modification-process-information has one argument of the sort design-process-state, and the relation is-included-in-which-design-process-states has two arguments of the sorts RQS-modification-process-info-element and sign, respectively.

An atom includes-which-RQS-modification-process-information(*design-process-state1*) specifies a query for the modification process information included in the process state *design-process-state1*. An atom is-included-in-which-design-process-states(*RQS-modification-process-info-element1*, *sign1*) specifies a query for the design process states that include the modification process information *RQS-modification-process-info-element1* with sign *sign1* as its truth value.

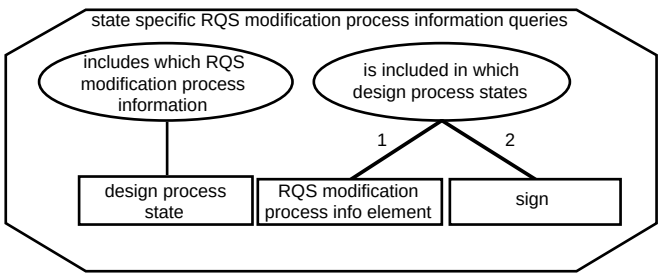


FIGURE 7.37. Structure of information type state specific RQS modification process information queries.

Example 7.27.
"There is currently a query for the modification process information included in the requirement qualification set manipulation process state designated State6. Furthermore, there is currently a query for the design process state in which it was decided to replace the current requirement qualification set by *Definition Study Doc v0.5*."

includes-which-RQS-modification-process-information(*State6*)
is-included-in-which-design-process-states(
  is-replacement-for-current-RQS("*Definition Study Doc v0.5*"), pos)

The information type application specific RQSM process state information queries models queries for application specific information about requirement qualification set manipulation process states. For example, a relation specified within this information type can be used to model queries for information about which manipulation process states are dead ends.

7.3 Relation between Compositions of Process and Knowledge

The relation between the process composition and the knowledge composition of an RQS manipulation process specifies how the processes involved in RQS manipulation relate to the reflection levels for a design process.

- At the first meta-level of a design process, the processes of current RQS maintenance and deductive RQS refinement reason with design requirement information (i.e., design requirement definitions and relations between design requirements).

- At the second meta-level of a design process, the processes of RQS manipulation history maintenance and RQS modification reason with requirement qualification sets and RQS assessments, as well as proposals for the modification of requirement qualifications sets.

- At the third meta-level of a design process, the processes of RQS manipulation history maintenance and RQS modification reason with overall design strategies, RQSM trace information and RQS manipulation process evaluations.

Figure 7.38 shows how the relation between process composition and knowledge composition of a requirement qualification set manipulation process is modelled in GDM.

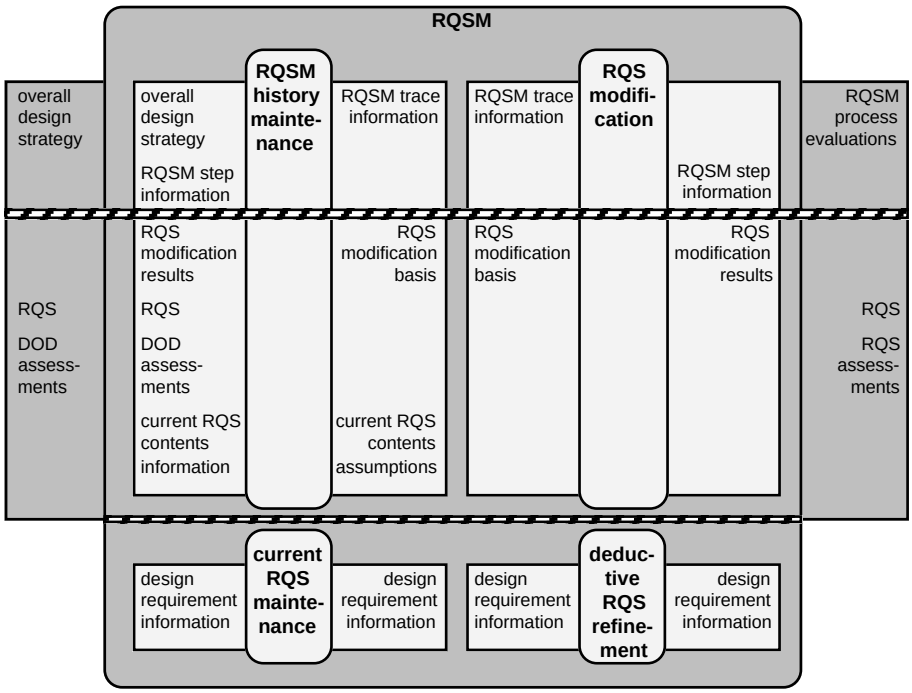


FIGURE 7.38. Relation between compositions of process and knowledge of an RQS manipulation process.

7.4 Use of the Model in Analysis and Development

This section explains how the model of a requirement qualification set manipulation process that GDM provides can be used in the analysis of practical design processes and the development of design support systems. Please note that the explanation given here should be read as an extension to Section 6.4, which focuses on the two highest levels of a design process.

Identification of requirement qualification set manipulation processes. During the analysis of applications, the following checklist can be used to determine whether or not a given application specific process is a requirement qualification set manipulation process.

- The input consists of an overall design strategy and a set of design requirements.
- The output consists of a set of design requirements.
- The intent is to follow the given overall design strategy in order to produce a set of design requirements that provides a solution to the given set of design requirements.

Application specific process composition. For any application specific requirement qualification set manipulation process, the sub-processes involved can be modelled by the components RQSM history maintenance, RQS modification, current RQS maintenance, and deductive RQS refinement. A modeller may decide to refine the components RQS modification and RQSM his-

tory maintenance to model the application specific modification of requirement qualification sets and maintenance of the history of the manipulation of requirement qualification sets. Chapter 9 describes possible refinements that have proven to be useful in many practical situations.

Application specific knowledge composition. The following information types within GDM can be used to model application specific objects and relations. (Note that in the detailed textual specification of GDM in Appendix B, these information types either have names that contain application specific as a prefix, or they include objects of which the names contain Example as a prefix.)

- The information type application specific RQSM process state information can be used to model application specific information about RQS manipulation process states, by means of relations on the sort design-process-state.
- The information type application specific RQSM process state information queries can be used to model queries for application specific information about RQS manipulation process states, by means of relations on the sort design-process-state.
- The information type RQS alteration type can be used to model alterations to the current requirement qualification set, by means of objects of the sort RQS-alteration.
- The information type application specific RQS alteration information can be used to model application specific information about alterations to the current requirement qualification set, by means of relations on the sorts RQS-name and RQS-alteration.

The knowledge base of the component deductive RQS refinement can be used to model application specific theories about design requirements.

Chapter 8

Design Object Description Manipulation

The most characteristic output of a design process is the description of an artefact, which includes information about the properties of the design object and about its relations with other objects in a design domain. The design object description manipulation process is responsible for generating, modifying and deductively refining design object descriptions. This chapter describes the model of design object description manipulation processes that GDM provides, which details the third level of process abstraction of a design process.

An essential aspect of design processes is the generation of a design object description that properly describes the behaviour, function, form and structure of a design object. A design object description is the most characteristic output of a design process.

For a design process to be considered successful, at some point of time a design object description must be generated that supports the qualified requirements of a specific requirement qualification set and that, at the same time, contains sufficient information for the intended use of the design object. (The latter means that the design object description is to be the basis for the assembly, construction, fabrication or some other form of implementation of the design object.) During a design process, there is usually no certainty as to whether or not such a design object description can be generated.

Firstly, there is usually no way of telling whether or not a design object description exists that can fulfil a given requirement qualification set. For most practical design problems, as argued in Chapters 3 and 4, there is a vast space of possible design object descriptions that must be explored. Except for small or well-known domains of application, there are usually

no design methods available that start with a given requirement qualification set and always terminate with a satisfactory design object description.

Secondly, as argued in Chapter 7, it may be inevitable that the design requirements take shape only during the design process. The initial requirement qualification set may be inconsistent, ambiguous, vague, and incomplete, which makes the generation of a satisfactory design object description impossible. Hence, it may be uncertain during the design process whether or not a satisfactory design object description can be generated.

During a design process, a *design object description manipulation process* modifies and deductively refines descriptions of the design object, where each description consists of information about the design object's properties and its relations with other domain objects. The goal is to deliver a consistent design object description that fulfils a given requirement qualification set and that contains sufficient domain object information for the intended use of the design object. Together with a design process co-ordination process and a requirement qualification set manipulation process, a design object description manipulation process forms a sub-process of a design process.

In Chapter 6, the two highest process abstraction levels of a design process have been described. This chapter describes the model of design object description manipulation processes that GDM provides, which details the third level of process abstraction of a design process. Section 8.1 presents the process composition of a design object description manipulation process, Section 8.2 the knowledge composition of a design object description manipulation process, and Section 8.3 the relation between process composition and knowledge composition of a design object description manipulation process. Finally, Section 8.4 explains how the model of a design object description manipulation process that GDM provides can be used in the analysis of practical design processes and the development of design support systems.

8.1 Process Composition

This section describes processes identified in design object description manipulation, and the composition of these processes.

8.1.1 Processes at different abstraction levels

This sub-section describes processes involved in design object description manipulation and the different levels of abstraction at which these processes play a role.

8.1.1.1 Processes

The following processes are involved at the two highest process abstraction levels of a design object description manipulation process:

- design object description manipulation (*DODM*),
- design object description modification (*DOD modification*),
- design object description manipulation history maintenance (*DODM history maintenance*),
- current design object description maintenance (*current DOD maintenance*),
- deductive design object description refinement (*deductive DOD refinement*).

In the following, these five processes are explained, together with the type of input information they use and the types of output information they produce. Note that design object description manipulation processes have already been introduced in Chapter 6; for the reader's convenience, parts of the explanation are repeated, and the same bicycle design example is used.

Design object description manipulation

A design object description manipulation process aims to generate a consistent design object description that fulfils a given requirement qualification set and that also includes sufficient domain object information for the intended use of the design object description. (Note that the intended use of a design object description is to be the basis for the assembly, construction, fabrication or other form of implementation of the design object.) This process always operates on a (possibly partial) description, called the *current design object description.*

The *current DODM process state* defines the current state of the design object description manipulation process. Only by well defined operations (such as the modification or deductive refinement of the current design object description) can the process as a whole make a transition to a new state, which then becomes the current state.

There are many methods that design object description manipulation may apply during a design process. For example, in a configuration process as described in Chapter 10, the task of design object description manipulation is to make a satisfactory configuration by selecting appropriate components and generating appropriate values for the parameters of these components. In an aircraft re-design process as described in Chapter 13, one method applied by design object description manipulation is to retrieve parts of an existing design object description generated for a similar, existing type of aeroplane that has been officially certified to be airworthy.

Figure 8.1 shows (on the left-hand side) the types of information that a design object description manipulation process uses as input and (on the right-hand side) the types of information that it produces as output. Refer to Chapter 6 for an example explaining the types of input and output information used by a design object description manipulation process.

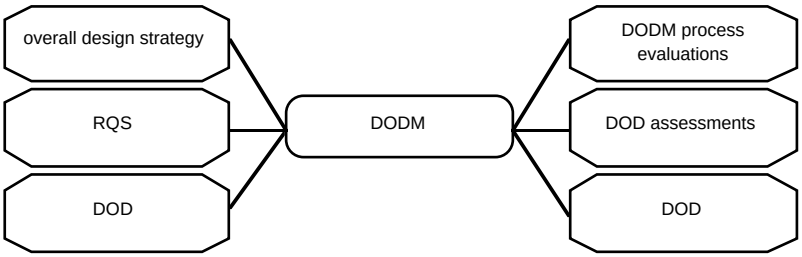


FIGURE 8.1. Input and output of a design object description manipulation process.

Design object description modification

A design object description modification process aims to modify the current design object description into a consistent description that fulfils the given requirement qualification set and that includes sufficient domain object information to construct a design object. This process acts on the basis of information about present and past states of the manipulation process, about steps (i.e., state transitions) taken within the manipulation process, about traces (i.e., sequences of steps) generated within the manipulation process, and about the (intermediate) results of the manipulation process. The process may modify the contents of the current design object description, or set a focus for deductive refinement of the current design object description. The process may also decide to retrieve an earlier generated design object description (to replace the current description), inspect the history of the design object description manipulation process (by expressing queries to be processed), or terminate the manipulation process.

Figure 8.2 shows the types of information that a design object description modification process uses as input and the types of information that it produces as output.

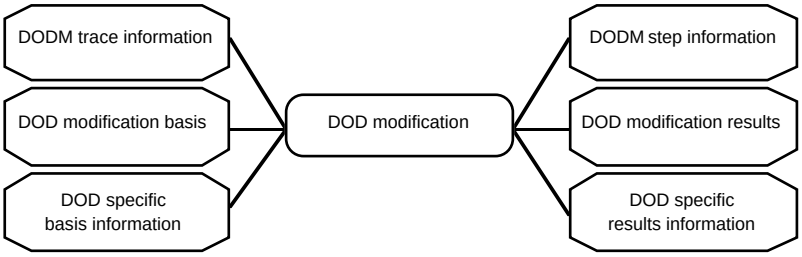


FIGURE 8.2. Input and output of a design object description modification process.

The following example, extending the bicycle design example of Chapter 6, is used to explain the types of input information used by a design object description modification process.

Example 8.1.
"The basis for modification of the current description named *Design Doc v0.3* consists of the information that the domain object information included in *Design Doc v0.3* is about a twelve-gear mountain bicycle that is safe to be used by young children and that costs 450 euro. Furthermore, *Design Doc v0.3* satisfies some, but not all of the design requirements included in the set named *Definition Study Doc v0.6*; for example, it does not satisfy the requirement that the bicycle must cost less than 400 euro.

In the design process, four design object description manipulation states have been generated so far. In the second state, the current description was the design object description named *Design Doc v0.1*, and the alteration proposal was selected to remove the shock absorbers from the bicycle's frame. In the current (fourth) process state, designated *State4*, an existing proposal is to make the wheels out of carbon, and a rejected proposal is to remove the shock absorbers from the bicycle's frame.

The decision taken in the first state was to deductively refine the current design, in the second state to modify the current design, and in the third state to retrieve an earlier generated design. The overall design strategy up to the second state was to generate a new bicycle design from scratch; this strategy failed. Since the third state, the overall design strategy is to use earlier bicycle design cases as a basis. Up to now, there has been no evaluation of this overall design strategy."

DOD modification basis. The *DOD modification basis* of a design object description modification process concerns basis information for preparing the next step of the design object description manipulation process. In GDM, the information type DOD modification basis can be used to model the DOD modification basis, including the following types of information:

- epistemic information about the following first meta-level information: domain object information included in specific design object descriptions, default domain object information applicable to specfic design object descriptions, and application specific information about design object descriptions,
- information about the design requirement information included in specific requirement qualification sets,
- design object description assessments,
- information about the identity of the current requirement qualification set,
- information about the identity of the current design object description,
- currently applicable proposals for alterations to the current design object description,
- currently applicable rejections of alterations to the current design object description,
- information about the composition of alterations to the current design object description,
- application specific information about alterations to the current design object description, and
- information about design object description solutions.

See Table 8.1 for the design object description modification basis in Example 8.1.

TABLE 8.1. DOD modification basis in Example 8.1.

Kind	*Content*
Epistemic DOD specific basis information	The basis for modification of the current description consists of the information that the domain object information included in the design *Design Doc v0.3* is about a twelve-gear mountain bicycle that is safe to be used by young children and that costs 450 euro.
Current DOD identity information	The current description is named *Design Doc v0.3*.
Proposed DOD alterations	An existing proposal is to make the wheels out of carbon.
Rejected DOD alterations	A rejected proposal is to remove the shock absorbers from the bicycle's frame.
RQS	The set of design requirements named *Definition Study Doc v0.6* includes, among others, the requirement that the bicycle must cost less than 400 euro.
DOD assessments	The design named *Design Doc v0.3* does not satisfy the requirement included in the set of design requirements named *Definition Study Doc v0.6* that the bicycle must cost less than 400 euro.

DODM trace information. A design object description modification process may need historical information about earlier states of the design object description manipulation process of which it is part, such as information about which alterations have already been tried, the sequence in which these alterations were made, what the results were, and so forth. The term *DODM trace information* denotes such information, which is modelled in GDM by the information type DODM trace information. Three types of DODM trace information are distinguished:

- design process state specific information about the modification of design object descriptions;
- generic information about the trace of the design object description manipulation process (e.g., information about the sequence of states in the manipulation process, the decisions for the steps made within the manipulation process, and the overall design strategy followed in a specific state);
- application specific information about individual states of the design object description manipulation process and their relations.

Note that DODM trace information in a design object description manipulation process has a similar composition as RQSM trace information in a requirement qualification set manipulation process. See Table 8.2 for part of the DODM trace information in Example 8.1.

TABLE 8.2. DODM trace information in Example 8.1.

Kind	*Content*
State specific DOD modification process information	In the second state, the proposal was selected to remove the shock absorbers from the bicycle's frame.
Manipulation trace information	There are four states so far. The decision in the first state was to deductively refine the current design, in the second state to modify the current design, and in the third state to retrieve an earlier generated design. The overall design strategy up to the second state was to generate a new bicycle design from scratch; this strategy failed. Since the third state, the overall design strategy is to use earlier bicycle design cases as a basis. Up to now, there has been no evaluation of this strategy.

The following example, extending Example 8.1, is used to explain the types of output information produced by a design object description modification process.

Example 8.2.
"Because the current design (named *Design Doc v0.3*) does not satisfy the requirement included in the set of design requirements named *Definition Study Doc v0.6* that the bicycle must cost less than 400 euro, it has been decided to modify the current design. The proposal to replace the disk brakes by ordinary brakes has been selected."

DOD modification results. With each step, a design object description modification process produces new (intermediate) results. The *DOD modification results* denote the modification information produced in the current design object description manipulation process state, which is modelled in GDM by the information type DOD modification results. The following types of DOD modification results are distinguished:

- epistemic information about the following first meta-level information: queries for information about the contents of design object descriptions, domain object information in focus for deductive refinement of the current design object description, default domain object information applicable to specific design object descriptions, and application specific information about design object descriptions,
- assessments of design object descriptions,
- information about the identity of a design object description to be retrieved,
- proposed alterations to the current design object description,
- selected alterations to the current design object description,
- information about the composition of alterations to the current description,
- application specific information about alterations to the current description, and
- information about design object description solutions.

These design object description modification results are optional as output of a design object description modification process. See Table 8.3 for the design object description modification results in Example 8.2.

TABLE 8.3. DOD modification results in Example 8.2.

Kind	*Content*
Selected DOD alterations	The proposal to replace the disk brakes by ordinary brakes has been selected.

DODM step information. Besides information related to modification results, a design object description modification process produces process-related information about the next step to be taken. The *DODM step information* of a design object description modification process denotes this type of information, which is modelled in GDM by the information type DODM step information. Five types of step information are distinguished:

- information about the decision for the next step of the design object description manipulation process (i.e., to retrieve an earlier generated description, to modify the current description, to deductively refine the current description, to inspect the history of the manipulation process, or to terminate the manipulation process),
- design object description manipulation process evaluations,
- queries for generic, design process state specific, information about the modification of design object descriptions,
- queries for generic information about the trace of the design object description manipulation process, and
- queries for application specific information about individual states of the design object description manipulation process and their relations.

Except for the current control decision, these types of information are optional as output of a design object description modification process. Note that DODM step information in a design object description manipulation process has a similar composition as RQSM step information in a requirement qualification set manipulation process. See Table 8.4 for part of the design object description manipulation step information in Example 8.2.

TABLE 8.4. DODM step information in Example 8.2.

Kind	*Content*
Current control decision information	It has been decided to modify the current design.

Design object description manipulation history maintenance

A design object description manipulation (DODM) history maintenance process aims to record the steps and results of a design object description manipulation process and to provide historical information that is of use in the current state of the manipulation process.

DODM history maintenance necessarily involves truth maintenance, to prevent the design object description modification process from making the same decisions (whether good or bad) twice for the same design object description and the same requirement qualification set to be fulfilled. That is, a modification proposal that has been selected earlier for the same design object description and the same requirement qualification set to be fulfilled should not be selected again (because it would lead to exactly the same results as produced earlier), and is marked rejected by the DODM history maintenance process.

Figure 8.3 shows the types of information that a DODM history maintenance process uses as input and the types of information that it produces as output.

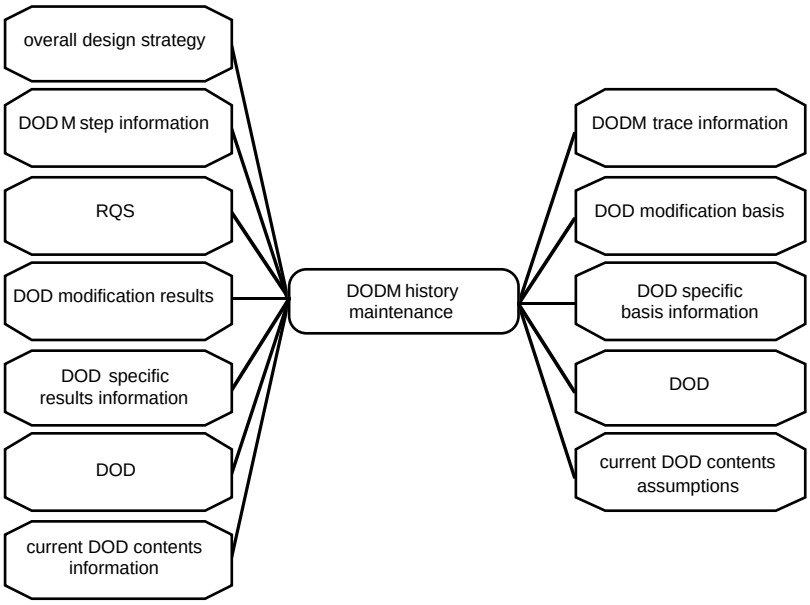


FIGURE 8.3. Input and output of a DODM history maintenance process.

The following example, extending Example 8.2, is used to explain the types of input information used by a DODM history maintenance process.

Example 8.3.

"The requirements included in the given set named *Definition Study Doc v0.6* are, among others, that the bicycle must be suited for riding in mountains and safe to be used by young children, and its price must be less than 400 euro. If the latter two requirements are irreconcilable, safety is preferred over price; the other requirements, though, must be satisfied. The design named *Design Doc v0.3* does not fulfil the set named *Definition Study Doc v0.6*. The decision has been taken to modify the current design, and the proposal to make the wheels out of carbon has been selected.

The overall design strategy is to use earlier bicycle design cases as a basis. The current design object description is known to include the information, among others, that the bicycle's wheels are to be made of carbon."

All of the input information types of a DODM history maintenance process have already been explained, except for current DOD contents information. This information type models epistemic information about the domain object information included in the current design object description. See Table 8.5 for the current DOD contents information in Example 8.3.

TABLE 8.5. Current DOD contents information in Example 8.3.

Kind	*Content*
Current DOD contents information	The current design object description is known to include the information, among others, that the bicycle's wheels are to be made of carbon.

The following example, extending Example 8.3, is used to explain the types of output information produced by a DODM history maintenance process.

Example 8.4.
"Assumptions for the current description are that it includes the information that the bicycle costs 350 euro and that the bicycle is able to carry a load of 160 kg. The design object description named *Design Doc v0.3* does not fulfil the set of design requirements named *Definition Study Doc v0.6*.

In the design process, five design object description manipulation states have been generated so far. The decision in the first state was to deductively refine the current design, in the second state to modify the current design, in the third state to retrieve an earlier generated design, and in the fourth state to modify the current design (which was, at that time, *Design Doc v0.3*). The current overall design strategy is to generate a design from scratch."

All of the output information types of a DODM history maintenance process have already been explained, except for current DOD contents assumptions. This information type models meta-level assumptions about the domain object information to be added to, or deleted from, the current design object description. See Table 8.6 for the current DOD contents assumptions in Example 8.4.

TABLE 8.6. Current DOD contents assumptions in Example 8.4.

Kind	*Content*
Current DOD contents assumptions	Assumptions for the current design are that it includes the information that the bicycle costs 350 euro and that the bicycle is able to carry a load of 160 kg.

Current design object description maintenance
A current design object description maintenance process aims to record the domain object information included in the current design object description and to provide up-to-date domain object information. The domain object information included in the current design object description may change due to the replacement of the current design object description by another description, the application of modifications to the current description, and the deductive refinement of the current description.

Figure 8.4 shows the type of information that a current design object description maintenance process uses as input and that it produces as output. This type has already been introduced in Chapter 6.

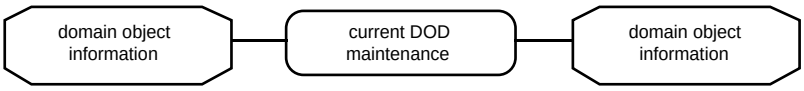


FIGURE 8.4. Input and output of a current design object description maintenance process.

Deductive design object description refinement
A deductive design object description refinement process deductively refines the current design object description on the basis of a domain-specific theory. Deductive refinement makes domain object information and relations between domain objects explicit that follow from the domain object information already available and the domain-specific theory.

Figure 8.5 shows the type of information that a deductive design object description refinement process uses as input and that it produces as output. This type has already been introduced in Chapter 6.

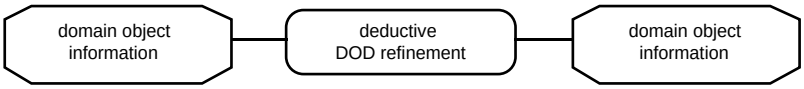


FIGURE 8.5. Input and output of a deductive design object description refinement process.

8.1.1.2 Process abstraction levels

A design object description manipulation process has been described to be composed of four processes: (1) a design object description modification process to determine appropriate modifications to the current design object description, (2) a DODM history maintenance process to maintain historical information about design object description manipulation, (3) a current design object description maintenance process to maintain the domain object information included in the current design object description, and (4) a deductive design object description refinement process to deductively refine the domain object information included in the current design object description.

This composition is modelled in GDM by four abstraction relations; Figure 8.6 shows these relations between the component DODM and the respective components DOD modification, DODM history maintenance, current DOD maintenance and deductive DOD refinement.

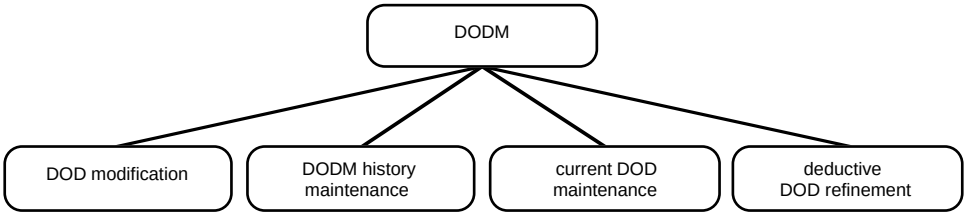


FIGURE 8.6. Two levels of abstraction for a DOD manipulation process.

8.1.2 Composition of processes

This section describes the way in which a design object description manipulation process is composed of lower-level processes, in terms of possibilities for exchange of information between processes, and in terms of task control knowledge used to control both the processes and the information exchange.

8.1.2.1 Information exchange

In a design object description manipulation process, all five processes involved (i.e., the process as a whole and its four sub-processes) exchange information. As explained in Chapter 5, a distinction is made between the information exchange between processes at different levels of process abstraction, and the information exchange between processes at the same level of process abstraction. Figure 8.7 gives a graphical overview of the exchange of information within a design object description manipulation process.

Firstly, a DOD manipulation process transfers the information it receives as input (the given overall design strategy, requirement qualification set and possibly a design object description) to its DOD manipulation history maintenance sub-process (see Table 8.7). This information transfer enables explicit recording of the input information with which the DOD manipulation process has been activated (in a new manipulation state to be generated by the DOD manipulation history maintenance process).

TABLE 8.7. Information transfer from a DOD manipulation process to its sub-processes.

Source	*Destination*	*Information link*	*Information type*
DODM	DODM history maintenance	Overall design strategy for DODM	Overall design strategy
DODM	DODM history maintenance	RQS for DODM	RQS
DODM	DODM history maintenance	DOD for DODM	DOD

Secondly, the four sub-processes of DOD manipulation exchange information with each other (see Table 8.8). DODM trace information is transferred from DODM history maintenance to DOD modification, as is the current DOD modification basis. If the current design object description has to be replaced by another description, current DOD contents assumptions are transferred from DODM history maintenance to current DOD maintenance.

Information about the next DODM step to be taken is transferred from DOD modification to DODM history maintenance, as are the current DOD modification results. A new focus for deductive refinement of the current design object description is transferred from DOD modification to deductive DOD refinement.

When it is to be stored, the current DOD contents information is transferred from current DOD maintenance to DODM history maintenance. When it is to be refined, the current domain object information is transferred from current DOD maintenance to deductive DOD refinement. Finally, results of refining the current design object description are transferred from deductive DOD refinement to current DOD maintenance.

TABLE 8.8. Information exchange between sub-processes of a DOD manipulation process.

Source	*Destination*	*Information link*	*Information type*
DODM history maintenance	DOD modification	Current DODM trace information	DODM trace information
DODM history maintenance	DOD modification	Current DOD modification basis	DOD modification basis
DODM history maintenance	Current DOD maintenance	Current DOD contents assumptions to be used	Current DOD assumptions
DOD modification	DODM history maintenance	Current DODM step information	DODM step information
DOD modification	DODM history maintenance	Current DOD modification results	DOD modification results
DOD modification	Deductive DOD refinement	Current deductive DOD refinement focus	Deductive DOD refinement focus
Current DOD maintenance	DODM history maintenance	Current DOD contents information to be used	Current DOD contents information
Current DOD maintenance	Deductive DOD refinement	Current domain object information	Domain object information
Deductive DOD refinement	Current DOD maintenance	Refined domain object information	Domain object information

Thirdly, part of the information that the four sub-processes of DOD manipulation produced as output is transferred to become output of the DOD manipulation process (see Table 8.9). DODM process evaluations are transferred from DOD modification, and appropriate design object descriptions and their assessments from DODM history maintenance.

TABLE 8.9. Information transfer to a DOD manipulation process from its sub-processes.

Source	*Destination*	*Information link*	*Information type*
DOD modification	DODM	Process evaluations from DODM	DODM process evaluations
DODM history maintenance	DODM	DOD from DODM	DOD
DODM history maintenance	DODM	DOD assessments from DODM	DOD assessments

The possibilities for information exchange between processes involved in DOD manipulation, as shown in Tables 8.7 to 8.9, are modelled in GDM by information links. Figure 8.7 shows a pictorial representation of the information links within the component DODM.

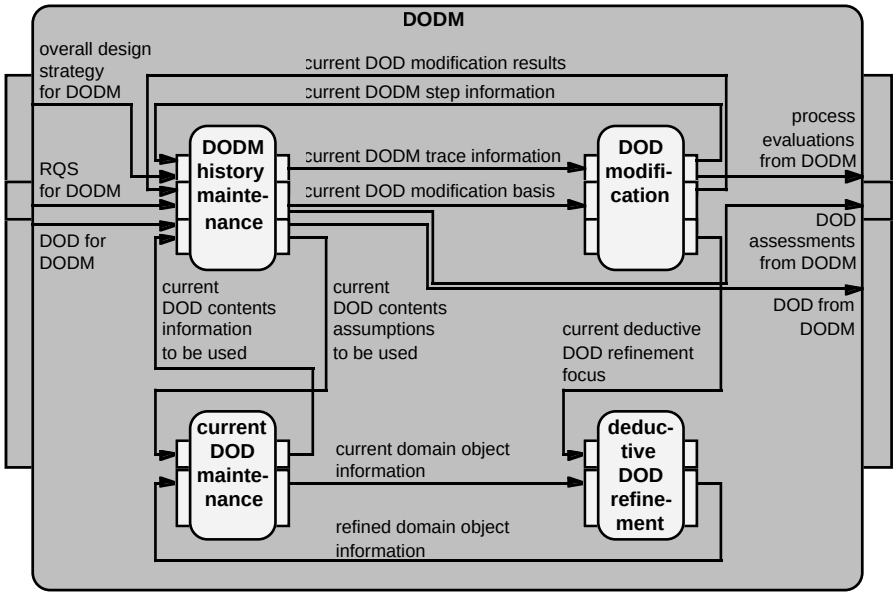


FIGURE 8.7. Information exchange between processes involved in DOD manipulation.

8.1.2.2 Task control

There are many ways in which a design object description manipulation process can be controlled. The generic control method described in this section can be used effectively in any design application, but it is not necessarily the most efficient. Note that in specific situations, other (more specific) control methods may be more suitable.

Start of a design object description manipulation process

When a design object description manipulation process starts, most likely receiving new input information, the following actions are taken:

- the given overall design strategy, requirement qualification set and design object description (if any) are transferred from the input of the DOD manipulation process to the input of DODM history maintenance,
- the task control focus of DODM history maintenance is set to be the commencement of the DOD manipulation process,
- DODM history maintenance is activated to store the received information.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if the component DODM is in its starting state, then in its next state the component DODM history maintenance will have been activated with the task control focus commencement, and the mediating links overall design strategy for DODM, RQS for DODM and DOD for DODM will have been updated.

Termination of DODM history maintenance
Which actions are to be taken when DODM history maintenance has terminated its activity depends on its task control focus. If DODM history maintenance has terminated and its task control focus is to commence the DOD manipulation process or to process deductive refinements of the current design object description, then the following actions are taken:

- the current DODM trace information and the current DOD modification basis are transferred from the output of DODM history maintenance to the input of DOD modification,
- DOD modification is activated to determine the next DOD manipulation step.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component DODM history maintenance is idle after having been active with task control focus commencement (denoting the start of the DOD manipulation process) or processing (denoting the act of processing deductive refinements of the current design object description), then in its next state the component DOD modification will have been activated, and the private links current DODM trace information and current DOD modification basis will have been updated.

If DODM history maintenance has terminated and its task control focus is to replace or to modify the current design object description, then the following actions are taken:

- assumptions about the domain object information included in the current design object description are transferred from the output of DODM history maintenance to the input of current DOD maintenance,
- current DOD maintenance is activated to update its domain object information.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component

DODM history maintenance is idle after having been active with task control focus replacement (denoting the act of replacing the current design object description) or modification (denoting the act of modifying the current design object description), then in its next state the component current DOD maintenance will have been activated and the private link current DOD contents assumptions will have been updated.

If DODM history maintenance has terminated and its task control focus is termination of the DOD manipulation process, then the following actions are taken:

- the current evaluations about the DOD manipulation process are transferred from the output of DOD modification to the output of the DOD manipulation process,
- the results of the DOD manipulation process (design object descriptions and their assessments) are transferred from the output of DODM history maintenance to the output of the DOD manipulation process,
- the DOD manipulation process is stopped.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component DODM history maintenance is idle after having been active with task control focus termination (denoting the termination of the DOD manipulation process), then in its next state the mediating links process evaluations from DODM, DOD from DODM and DOD assessments from DODM will have been updated and the component DODM will have stopped.

Termination of DOD modification

Which actions are to be taken when DOD modification has terminated its activity depends on which evaluation criterion has succeeded. The possible evaluation criteria refer to the following DOD manipulation events: (1) termination of the DOD manipulation process, (2) inspection of the DOD manipulation history, (3) replacement of the current design object description, (4) modification of the current design object description, and (5) deductive refinement of the current design object description.

If DOD modification has terminated and its evaluation criterion to terminate the DOD manipulation process has succeeded, then the following actions are taken:

- the current DODM step information and the current results of DOD modification are transferred from the output of DOD modification to the input of DODM history maintenance,
- the task control focus of DODM history maintenance is set to *termination*,
- DODM history maintenance is activated to store the current DOD modification results and DODM step information.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component

DOD modification is idle after having been active, and its evaluation criterion termination succeeds (denoting the termination of the DOD manipulation process), then in its next state the task control focus of the component DODM history maintenance will have been set to termination, the component DODM history maintenance will have been activated and the private links current DODM step information and current DOD modification results will have been updated.

If DOD modification has terminated and its evaluation criterion to query and retrieve the DOD manipulation history has succeeded, then the following actions are taken:

- the current DODM step information and the current results of DOD modification (including queries for historical information) are transferred from the output of DOD modification to the input of DODM history maintenance,
- the task control focus of DODM history maintenance is set to *query and retrieval*,
- DODM history maintenance is activated to store the current DOD modification results and DODM step information.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component DOD modification is idle after having been active, and its evaluation criterion query-and-retrieval succeeds (denoting that the DOD manipulation history is to be inspected), then in its next state the task control focus of the component DODM history maintenance will have been set to query-and-retrieval, the component DODM history maintenance will have been activated and the private links current DODM step information and current DOD modification results will have been updated.

If DOD modification has terminated and its evaluation criterion to replace the current design object description has succeeded, then the following actions are taken:

- the current DODM step information and the current results of DOD modification are transferred from the output of DOD modification to the input of DODM history maintenance,
- the task control focus of DODM history maintenance is set to *replacement*,
- DODM history maintenance is activated to store the current DOD modification results and DODM step information.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component DOD modification is idle after having been active, and its evaluation criterion replacement succeeds (denoting that the current design object description is to be replaced), then in its next state the task control focus of the component DODM history maintenance will have been set to replacement, the component DODM history maintenance will have been activated and the private links current DODM step information and current DOD modification results will have been updated.

If DOD modification has terminated and its evaluation criterion to modify the current design object description has succeeded, then the following actions are taken:

- the current DODM step information and the current results of DOD modification are transferred from the output of DOD modification to the input of DODM history maintenance,
- the task control focus of DODM history maintenance is set to *modification*,
- DODM history maintenance is activated to store the current DOD modification results and DODM step information.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component DOD modification is idle after having been active, and its evaluation criterion modification succeeds (denoting that the current design object description is to be modified), then in its next state the task control focus of the component DODM history maintenance will have been set to modification, the component DODM history maintenance will have been activated, and the private links current DODM step information and current DOD modification results will have been updated.

If DOD modification has terminated and its evaluation criterion to deductively refine the current design object description has succeeded, then the following actions are taken:

- the current DODM step information and the current results of DOD modification are transferred from the output of DOD modification to the input of DODM history maintenance,
- the current deductive DOD refinement focus is transferred from the output of DOD modification to the input of deductive DOD refinement,
- the current domain object information is transferred from the output of current DOD maintenance to the input of deductive DOD refinement,
- deductive DOD refinement is activated to refine the current design object description on the basis of the given refinement focus.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component DOD modification is idle after having been active, and its evaluation criterion deductive-refinement succeeds (denoting that the current design object description is to be deductively refined), then in its next state the task control focus of the component DODM history maintenance will have been set to deductive-refinement, the component deductive DOD refinement will have been activated, and the private links current DODM step information, current DOD modification results, current domain object information and current deductive DOD refinement focus will have been updated.

Termination of current DOD maintenance

Which actions are to be taken when current DOD maintenance has terminated its activity depends on which evaluation criterion of DOD modification has succeeded. The evaluation criteria to be considered are: (1) retrieval of an earlier generated design object description, (2) modification of the current design object description, and (3) deductive refinement of the current design object description.

If current DOD maintenance has terminated and the evaluation criterion of DOD modification to retrieve an earlier generated design object description or to modify the current design object description has succeeded, then the following actions are taken:

- the current DODM trace information and the current DOD modification basis are transferred from the output of DODM history maintenance to the input of DOD modification,
- DOD modification is activated to determine the next DOD manipulation step.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component current DOD maintenance is idle after having been active and the evaluation criterion replacement (denoting the act of replacing the current design object description) or the evaluation criterion modification (denoting the act of modifying the current design object description) of the component DOD modification has succeeded, then in its next state the component DOD modification will have been activated, and the private links current DODM trace information and current DOD modification basis will have been updated.

If current DOD maintenance has terminated and the evaluation criterion of DOD modification to deductively refine the current design object description has succeeded, then the following actions are taken:

- the epistemic information about the contents of the deductively refined current design object description is transferred from the output of current DOD maintenance to the input of DODM history maintenance,
- the task control focus of RQSM history maintenance is set to *processing*,
- DODM history maintenance is activated to store the epistemic information about the contents of the current design object description.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component current DOD maintenance is idle after having been active and the evaluation criterion deductive-refinement (denoting the act of deductively refining the current design object description) of the component DOD modification has succeeded, then in its next state the component DODM history maintenance will have been activated and the private link current DOD contents information to be used will have been updated.

Termination of deductive DOD refinement
If deductive DOD refinement has terminated, then the following actions are taken:

- the domain object information included in the deductively refined current design object description is transferred from the output of deductive DOD refinement to the input of current DOD maintenance,
- current DOD maintenance is activated to store the given domain object information.

This control knowledge is modelled in GDM by task control knowledge within the component DODM, declaring that if, in the current state of the component DODM, the component deductive DOD refinement is idle after having been active, then in its next state the component current DOD maintenance will have been activated, and the private link refined domain object information will have been updated.

8.2 Knowledge Composition

This section describes knowledge structures identified in design object description manipulation at different levels of abstraction, and the composition of these knowledge structures.

8.2.1 Reflection levels

In a design object description manipulation process, the following four reflection levels are distinguished:

- The *object level* includes information about domain objects. This level also includes deductive knowledge to derive implicit domain object information.

- The *first meta-level*, directly above the object level, includes information about design object descriptions. This level also includes deductive knowledge to derive implicit relations between design object descriptions and to derive defaults for domain object information (which may be used to supplement partial domain object information).

- The *second meta-level,* directly above the first meta-level, includes information about requirement qualification sets and about the relations between specific design object descriptions and requirement qualification sets. This level also includes knowledge to assess design object descriptions, as well as strategic knowledge to determine possible modifications of design object descriptions.

- The *third meta-level*, directly above the second meta-level, includes information about overall design strategies and about the trace and process evaluations of a design object

description manipulation process. This level also includes knowledge to determine strategies for the manipulation of design object descriptions.

8.2.2 Knowledge structures and composition

This section describes the composition and structure of the generic knowledge related to design object description manipulation, and it shows how they are modelled in GDM. For the sake of comprehension, the boxes with thick lines in the figures indicate the application specific knowledge structures of GDM that are candidates for refinement when using GDM to model a design object description manipulation process in a specific application domain.

The presented examples extend the bicycle design example. In these examples, terms and expressions shown in italics are part of the model of the application domain, not of GDM.

8.2.2.1 Structure and composition of knowledge at the object level

The object level includes *domain object information*, describing design artefacts, other domain objects, and their relations. Refer to Chapter 6 for the composition and structure of the information type domain object information.

8.2.2.2 Structure and composition of knowledge at the first meta-level

The first meta-level includes the following main types of information: epistemic information as well as assumptions about the contents of the current design object description, and information about specific design object descriptions, as used or produced by a design object description modification process.

Current DOD contents information. Figure 8.8 shows the composition of the information type current DOD contents information, which models epistemic information about the contents of the current design object description.

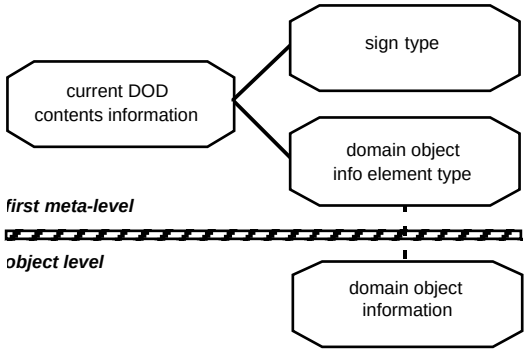


FIGURE 8.8. Composition of information type current DOD contents information.

This information type is used in situations where the contents of the current design object description have changed (due to modification or deductive refinement; in these situations, the new contents have to be reflected upwards, included in a new design object description, and stored as part of the manipulation history.

Figure 8.9 shows the structure of the information type current DOD contents information. It contains the relation is-currently-included, which has two arguments of the sorts domain-object-info-element and sign, respectively. An atom is-currently-included(*domain-object-info-element1*, *sign1*) specifies that the domain object information *domain-object-info-element1*, with sign *sign1* as its truth value, is known to be included in the current design object description.

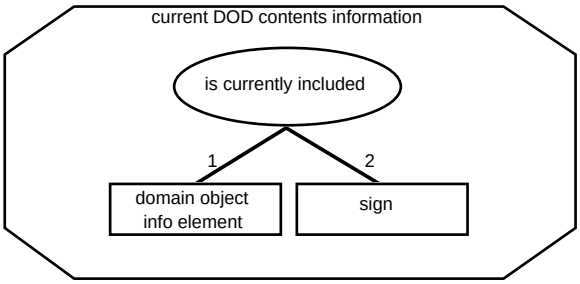


FIGURE 8.9. Structure of information type current DOD contents information.

Example 8.5.
"The domain object information included in the current description is that the bicycle has a six-gear motion-transfer system."

is-currently-included(has-part(*bicycle1*, *motion-transfer-system1*), pos)
is-currently-included(has-value(*motion-transfer-system1*, *number-of-gears*, 6), pos)

DOD specific basis information. Figure 8.10 shows the composition of the information type DOD specific basis information, which models information about specific design object descriptions, used by the process of modifying the current design object description. Refer to Chapter 6 for the composition and structure of the information types design requirement information, basic DOD assessments, domain object information, domain object info element type, sign type, DOD name type, and DOD.

Figure 8.11 shows the structure of the information type default domain object information, which models default domain object information applicable to a specific design object description. In practice, this default information is most often based on heuristics available for the application domain and may be used in a modification of the current design object description in order to supplement partial domain object information.

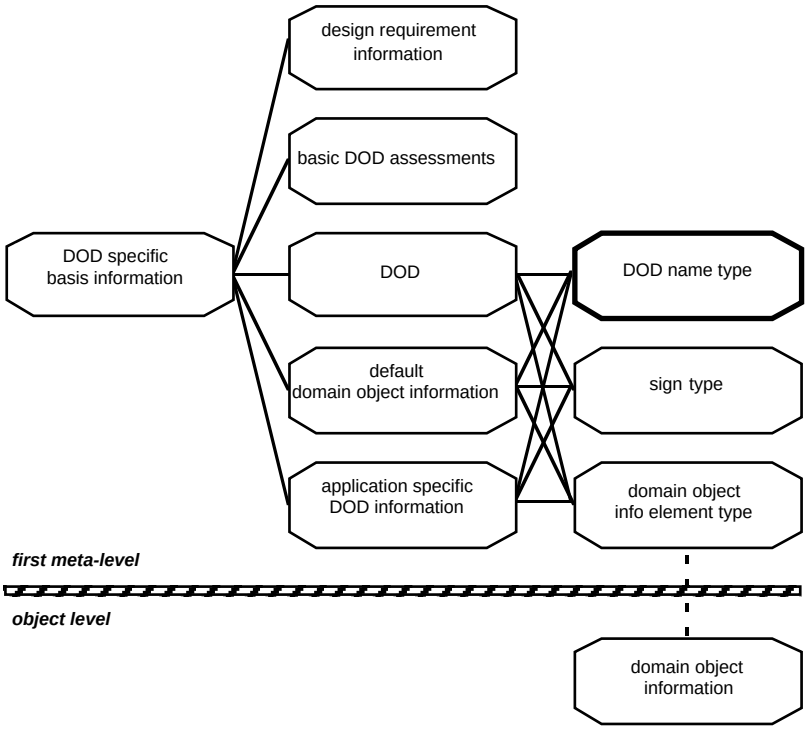


FIGURE 8.10. Composition of information type DOD specific basis information.

The information type default domain object information contains the relation has-default-domain-object-information, which has three arguments of the sorts DOD-name, domain-object-info-element, and sign, respectively. An atom has-default-domain-object-information(*DOD-name1*, *domain-object-info-element1*, *sign1*) specifies that the domain object information *domain-object-info-element1*, with *sign1* as its truth value, applies as a default to the design object description named *DOD-name1*.

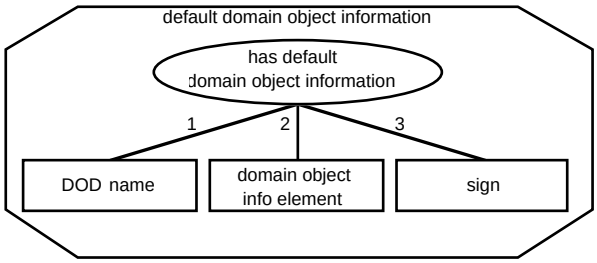


FIGURE 8.11. Structure of information type default domain object information.

Example 8.6.
"A default for the design named *Design v1* is that the bicycle has two wheels."

has-default-domain-object-information("*Design v1*", has-value(*bicycle1*, *num-wheels*, 2), pos)

The information type application specific DOD information models application specific information about specific design object descriptions. For example, in the domain of computer software, the fact that a specific design object description represents a functional design can be modelled by means of a relation specified within this information type.

Current DOD contents assumptions. Figure 8.12 shows the composition of the information type current DOD contents assumptions, which models assumptions about the contents of the current design object description.

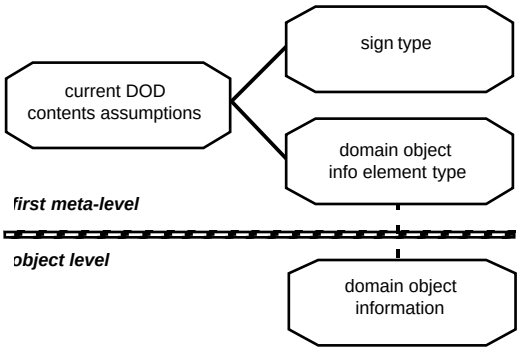


FIGURE 8.12. Composition of information type current DOD contents assumptions.

Figure 8.13 shows the structure of the information type current DOD contents assumptions. It contains two relations, is-to-be-asserted and is-to-be-retracted, which both have two arguments of the sorts domain-object-info-element and sign, respectively. An atom is-to-be-asserted(*domain-object-info-element1*, *sign1*) specifies that the domain object information *domain-object-info-element1*, with sign *sign1* as its truth value, is assumed to be included in the current design object description. An atom is-to-be-retracted(*domain-object-info-element1*, *sign1*) specifies that the domain object information *domain-object-info-element1*, with sign *sign1* as its truth value, is assumed not to be included in the current design object description.

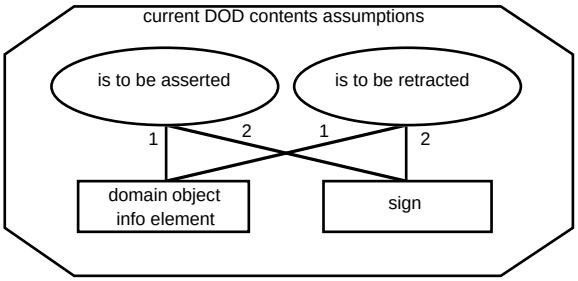


FIGURE 8.13. Structure of information type current DOD contents assumptions.

Example 8.7.
"The current description is assumed to include the fact that the domain object information that the bicycle has a six-gear motion-transfer system, and it is assumed not to include the fact that the wheels are made of carbon."

is-to-be-asserted(has-part(*bicycle1*, *motion-transfer-system1*), pos)
is-to-be-asserted(has-value(*motion-transfer-system1*, *number-of-gears*, 6), pos)
is-to-be-retracted(has-value(*front-wheel1*, *material*, *carbon*), pos)
is-to-be-retracted(has-value(*back-wheel1*, *material*, *carbon*), pos)

DOD specific results information. Figure 8.14 shows the composition of the information type DOD specific results information, which models information about specific design object descriptions, produced by the process of modifying the current design object description.

The information types basic DOD assessments, domain object information, sign type, domain object info element type, DOD name type, default domain object information, and application specific DOD information have already been explained. In the following, the other information types are explained to which the information type DOD specific results information refers.

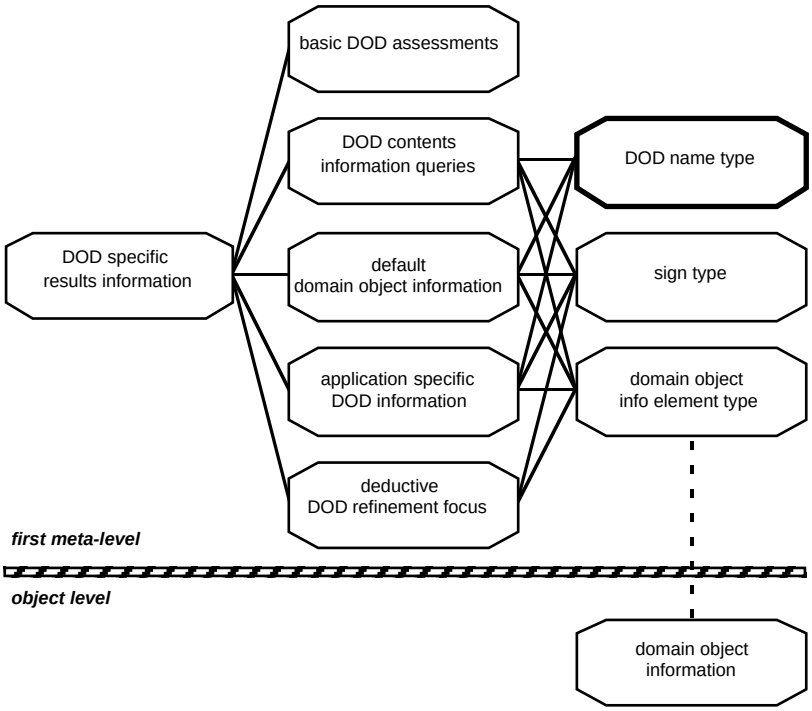


FIGURE 8.14. Composition of information type DOD specific results information.

Figure 8.15 shows the structure of the information type deductive DOD refinement focus, which models information about the current focus for the deductive refinement of the current design object description. It contains one relation, is-part-of-deductive-DOD-refinement-focus, which has two arguments of the sorts domain-object-info-element and sign, respectively. An atom is-part-of-deductive-DOD-refinement-focus(*domain-object-info-element1*, *sign1*) specifies that the domain object information *domain-object-info-element1*, with sign *sign1* as its truth value, is part of the current focus for deductive refinement of the current design object description.

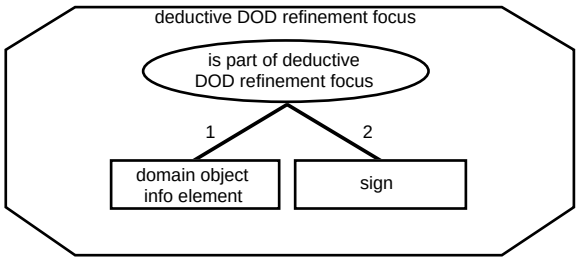


FIGURE 8.15. Structure of information type deductive DOD refinement focus.

Example 8.8.
"Of current interest are properties such as the weight that the bicycle is able to carry."

is-part-of-deductive-DOD-refinement-focus(*has-tolerance*(*bicycle1*, *load*, *weight*(*N*, *kg*)), pos)

Figure 8.16 shows the structure of the information type DOD contents information queries, which models queries for information about the contents of design object descriptions. It contains two relations: the relation includes-which-domain-object-information has one argument of the sort DOD-name, and the relation is-included-in-which-DODs has two arguments of the sorts domain-object-info-element and sign, respectively.

An atom includes-which-domain-object-information(*DOD-name1*) specifies a query for the domain object information included in the design object description named *DOD-name1*. An atom is-included-in-which-DODs(*domain-object-info-element1*, *sign1*) specifies a query for the design object descriptions that include the domain object information *domain-object-info-element1* with sign *sign1* as its truth value.

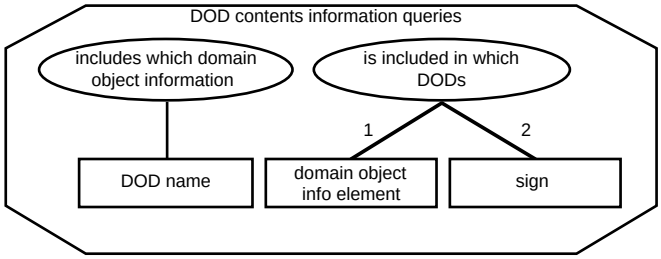


FIGURE 8.16. Structure of information type DOD contents information queries.

Example 8.9.
"There is currently a query for the domain object information included in the design named *Design Doc v0.1*. There is also a query for the design object descriptions that include the information that the bicycle is able to carry a load of 160 kg."

includes-which-domain-object-information("*Design Doc v0.1*")
is-included-in-which-DODs(*has-tolerance*(*bicycle1*, *load*, *weight*(160, *kg*)), pos)

8.2.2.3 Structure and composition of knowledge at the second meta-level

The second meta-level includes information used as a basis for modification of the current design object description, as well as information resulting from the process of modifying the current design object description.

DOD modification basis. Figure 8.17 shows the composition of the information type DOD modification basis, which models information that is used as a basis for modification of the current design object description. This information, together with DODM trace information (explained in the next sub-section), forms the input of the component DOD modification.

Refer to Chapter 6 for the composition and structure of the information types RQS, DOD assessments, and domain object info element type. Refer to Chapter 7 for the composition and structure of the information type current RQS identity information. In the following, the other information types are explained to which the information type DOD modification basis refers.

A *DOD modification* specifies either the addition of domain object information to the current design object description, or the deletion of domain object information from the current design object description. Objects of the sort DOD-modification within the information type DOD modification type are used to model DOD modifications. This information type specifies the function domain-object-information, which has two arguments of the sorts domain-object-info-element and sign, respectively, and which has the sort domain-object-information-literal as its range. This function is used to model domain object information that may be added to, or deleted from, the current design object description.

This information type also specifies two functions addition-of and deletion-of on the sort DOD-modification, which both have one argument of the sort domain-object-info-element. The function addition-of is used to model the addition of specific domain object information to the current design object description, whereas the function deletion-of is used to model the deletion of specific domain object information from the current design object description.

A *DOD alteration* specifies a set of one or more modifications to be made to the current design object description. Objects of the sort DOD-alteration within the information type DOD alteration type are used to model DOD alterations. An alteration may be formulated as a single DOD modification (modelled by an object of the sub-sort DOD-modification).

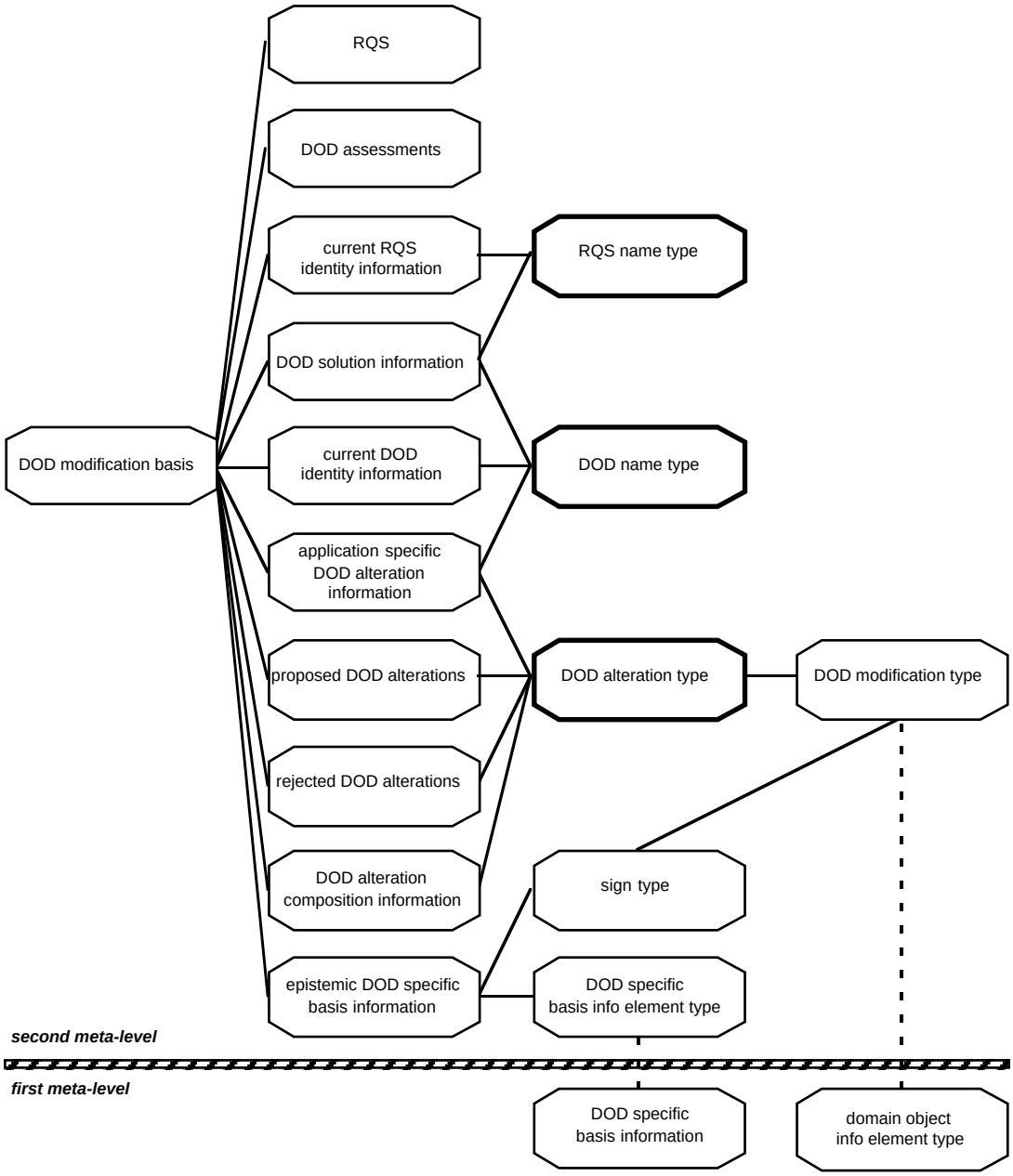


FIGURE 8.17. Composition of information type DOD modification basis.

Figure 8.18 shows the structure of the information type current DOD identity information, which models information about the identity of the current design object description. It contains the relation is-current-DOD, which has one argument of the sort DOD-name. An atom is-current-DOD(*DOD-name1*) specifies that the design object description named *DOD-name1* is the current design object description (and subject to modification by the design object description modification process).

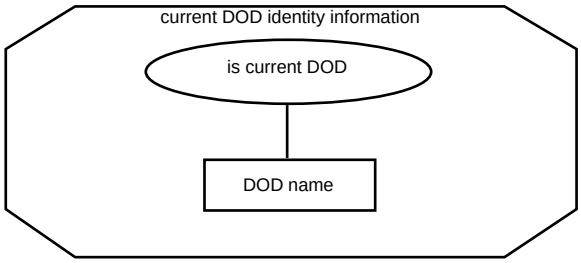


FIGURE 8.18. Structure of information type current DOD identity information.

Example 8.10.
"The current description is named *Design Doc v0.4*."

is-current-DOD("*Design Doc v0.4*")

The information type DOD specific basis info element type models meta-descriptions of information about specific design object descriptions, used by the process of modifying the current design object description. These meta-descriptions are obtained by the upward reflection of basis information (modelled by the information type DOD specific basis information) and are used to formulate epistemic information.

Figure 8.19 shows the structure of the information type epistemic DOD specific basis information, which models epistemic information about specific design object descriptions, used by a design object description modification process. It contains one relation, is-part-of-DOD-modification-basis, which has two arguments of the sorts DOD-specific-basis-info-element and sign, respectively. An atom is-part-of-DOD-modification-basis(*DOD-specific-basis-info-element1*, *sign1*) specifies that the basis information *DOD-specific-basis-info-element1*, with sign *sign1* as its truth value, is part of the basis for modification of the current design object description.

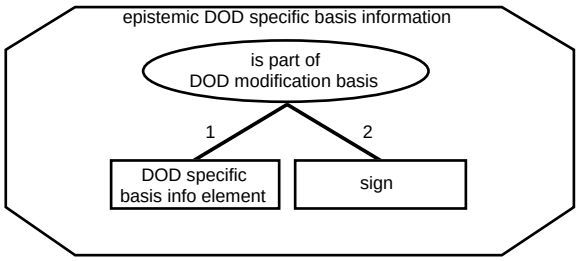


FIGURE 8.19. Structure of information type epistemic DOD specific basis information.

Example 8.11.
"The basis information for modification of the current description includes the fact that a default for the design named *Design v1* is that the bicycle has two wheels."

is-part-of-current-DOD-modification-basis(has-default-domain-object-information(
"*Design v1*", has-value(*bicycle1*, *num-wheels*, 2), pos), pos)

The information type application specific DOD alteration information models application specific information about design object description alterations. For example, a relation specified within this information type can be used to model preferences for different alterations.

Figure 8.20 shows the structure of the information type DOD alteration composition information, which models information about the modifications included in a specific alteration to the current design object description. It contains the relation includes-DOD-modification, which has two arguments of the sorts DOD-alteration and DOD-modification, respectively. An atom includes-DOD-modification(*DOD-alteration1*, *DOD-modification1*) specifies that the alteration *DOD-alteration1* includes the modification *DOD-modification1*.

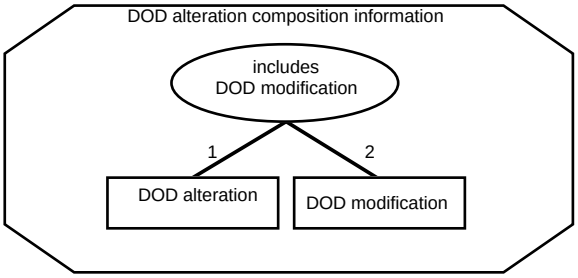


FIGURE 8.20. Structure of information type DOD alteration composition information.

Example 8.12.
"One possibility (option 1) to alter the current description is to replace the property that the bicycle is low priced by a property that the bicycle's price is 350 euro."

includes-DOD-modification(*AlterationOption1*, deletion-of(*is-price-level*(*bicycle1*, *low*)))
includes-DOD-modification(*AlterationOption1*,
addition-of(has-value(*bicycle*, *price*(*euro*), 350)))

Figure 8.21 shows the structure of the information type proposed DOD modifications, which models information about proposed alterations to the current design object description. It contains the relation is-proposed-DOD-alteration, which has one argument of the sort DOD-alteration. An atom is-proposed-DOD-alteration(*DOD-alteration1*) specifies that the alteration *DOD-alteration1* is an existing proposal to modify the current design object description.

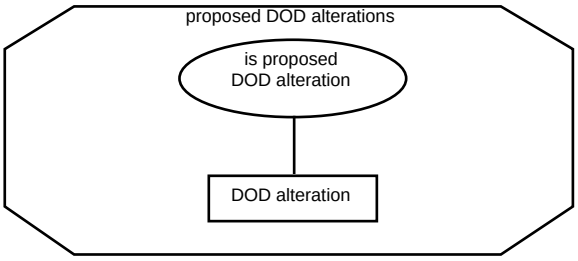


FIGURE 8.21. Structure of information type proposed DOD alterations.

Example 8.13.
"A current proposal (option 1) is to replace the property that the bicycle is low priced by a property that the bicycle's price is 350 euro."

is-proposed-DOD-alteration(*AlterationOption1*)

Figure 8.22 shows the structure of the information type rejected DOD alterations, which models information about rejected alterations to the current design object description. (These alterations have been applied earlier to the same design object description for the same requirement qualification set to be fulfilled, so to prevent them from being applied again, they have been explicitly marked as rejected.) The information type contains the relation is-rejected-DOD-alteration, which has one argument of the sort DOD-alteration. An atom is-rejected-DOD-alteration(*DOD-alteration1*) specifies that the alteration *DOD-alteration1* is rejected as a proposal to modify the current design object description.

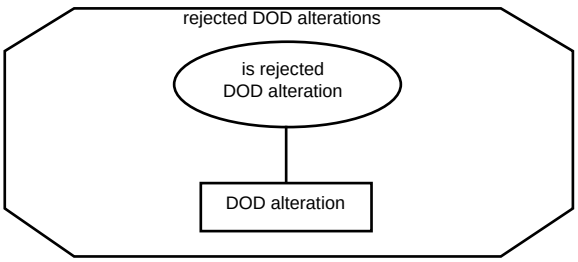


FIGURE 8.22. Structure of information type rejected DOD alterations.

Example 8.14.
"The proposal to remove the shock absorbers from the bicycle's frame has been rejected."

is-rejected-DOD-alteration(
deletion-of(has-value(*frame1*, *shock-absorbers*, [*shock-absorber1*, *shock-absorber2*]), pos)))

DOD modification results. Figure 8.23 shows the composition of the information type DOD modification results, which models information resulting from the process of modifying the current design object description. This information, together with DODM step information (explained in the next sub-section), forms the output of the component DOD modification.

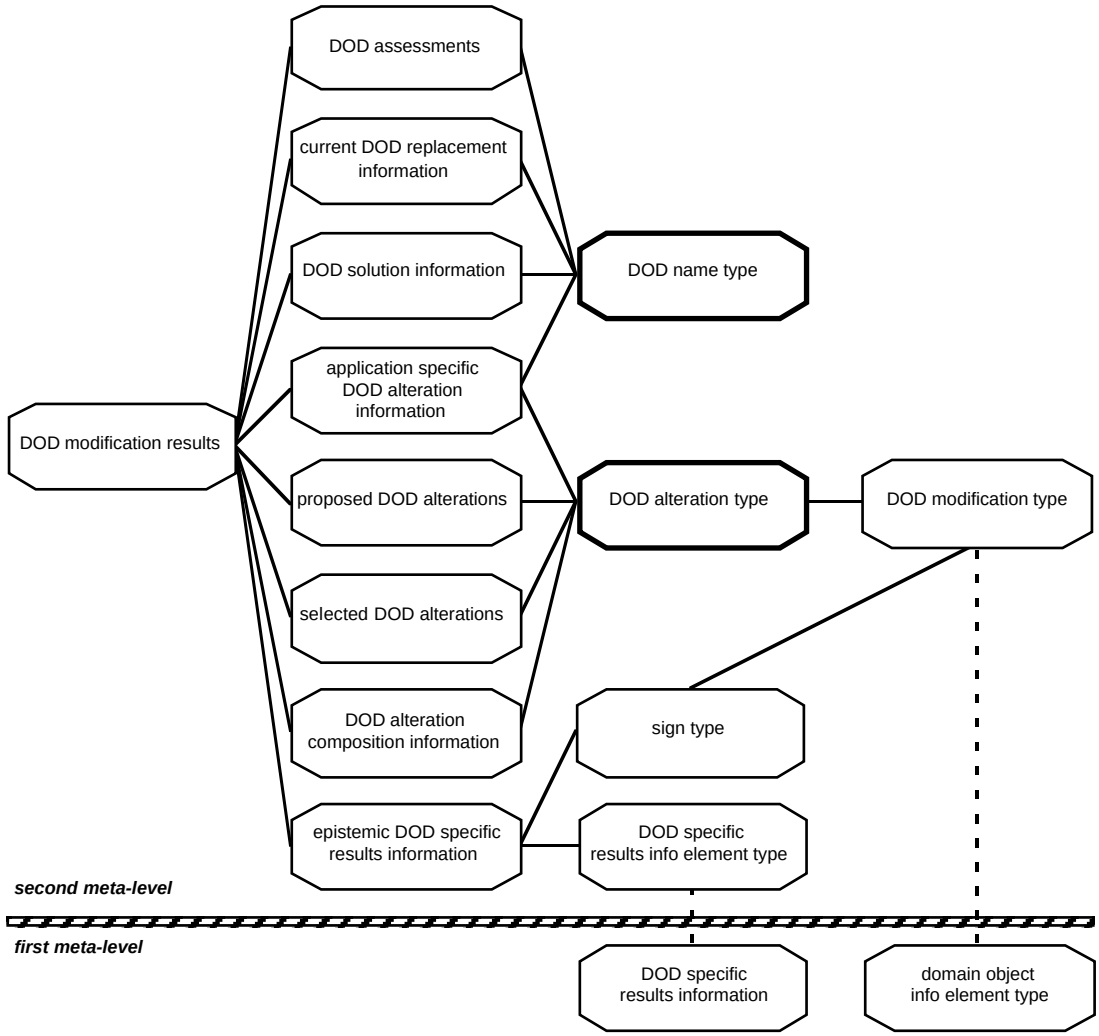


FIGURE 8.23. Composition of information type DOD modification results.

The information types domain object info element type, sign type, DOD name type, RQS name type, DOD assessments, DOD solution information, DOD modification type, DOD alteration type, DOD alteration composition information, proposed DOD alterations, and application specific DOD alteration information have already been explained. In the following, the other information types are explained to which the information type DOD modification results refers.

Figure 8.24 shows the structure of the information type current DOD replacement information, which models information about the identity of the design object description that is to replace the current design object description. It contains the relation is-replacement-for-current-DOD, which has one argument of the sort DOD-name. An atom is-replacement-for-current-DOD(*DOD-name1*) specifies that the design object description named *DOD-name1* is to replace the current design object description.

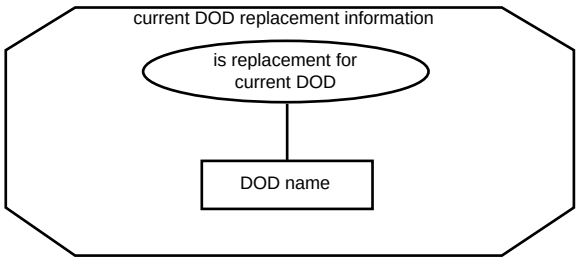


FIGURE 8.24. Structure of information type current DOD replacement information.

Example 8.15.
"The current description has to be replaced by the design named *Design Doc v0.2*."

is-replacement-for-current-DOD("*Design Doc v0.2*")

The information type DOD specific results info element type models meta-descriptions of information about specific design object descriptions, produced by the process of modifying the current design object description. These meta-descriptions are obtained by the upward reflection of results information (modelled by the information type DOD specific results information) and are used to formulate epistemic information.

Figure 8.25 shows the structure of the information type epistemic DOD specific results information, which models epistemic information about specific design object descriptions, produced by the process of modifying the current design object description. It contains one relation, is-part-of-DOD-modification-results, which has two arguments of the sorts DOD-specific-results-info-element and sign, respectively.

An atom is-part-of-DOD-modification-results(*DOD-specific-results-info-element1*, *sign1*) specifies that the results information *DOD-specific-results-info-element1*, with sign *sign1* as its truth value, is part of the results of modifying the current design object description.

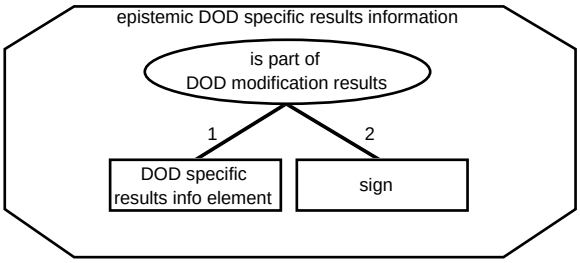


FIGURE 8.25. Structure of information type epistemic DOD specific results information.

Example 8.16.
"The resulting information from the process of modifying the current description includes that a default for the design named *Design Doc v0.4* is that the bicycle's frame is made of metal."

is-part-of-current-DOD-modification-results(has-default-domain-object-information(
"*Design Doc v0.4*", has-value(*frame1*, *material*, *metal*), pos), pos)

Figure 8.26 shows the structure of the information type selected DOD alterations, which models information about selected alterations to the current design object description. It contains the relation is-selected-DOD-alteration, which has one argument of the sort DOD-alteration. An atom is-selected-DOD-alteration(*DOD-alteration1*) specifies that the alteration *DOD-alteration1* is selected (i.e., this alteration is to be applied in order to modify the current design object description).

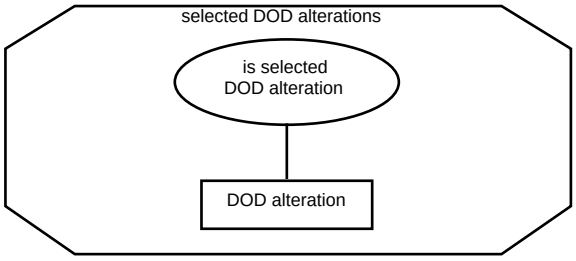


FIGURE 8.26. Structure of information type selected DOD alterations.

Example 8.17.
"The proposal (option 1) is selected to replace the property that the bicycle is low priced by a property that the bicycle's price is 350 euro."

is-selected-DOD-alteration(*AlterationOption1*)

DOD modification process information. Figure 8.27 shows the composition of the information type DOD modification process information, which models information about the input and output of a DOD modification process. It refers to two information types, DOD modification basis and DOD modification results, which together model the basis for DOD modification and the results of DOD modification. (This information type is used to reflect DOD modification process information upward to the third meta-level.)

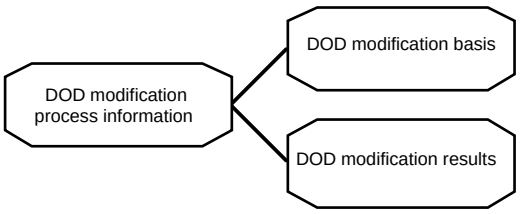


FIGURE 8.27. Composition of information type DOD modification process information.

8.2.2.4 Structure and composition of knowledge at the third meta-level

The third meta-level includes information about traces and steps of DOD manipulation processes.

DODM trace information. Figure 8.28 shows the composition of the information type DODM trace information, which models information about the trace of the design object description manipulation process.

Refer to Chapter 7 for the composition and the structure of the information type manipulation trace information. In the following, the other information types are explained to which the information type DODM trace information refers.

The information type DOD modification process info element type models meta-descriptions of information about the process of modifying the current design object description. These meta-descriptions are obtained by the upward reflection of process information about design object description modification (modelled by the information type DOD modification process information) and are used to formulate epistemic information.

The information type application specific DODM process state information models application specific information about different states of a design object description manipulation process. For example, a relation within this information type can be used to model the heuristics-based information that a specific design object description manipulation process state is a dead end.

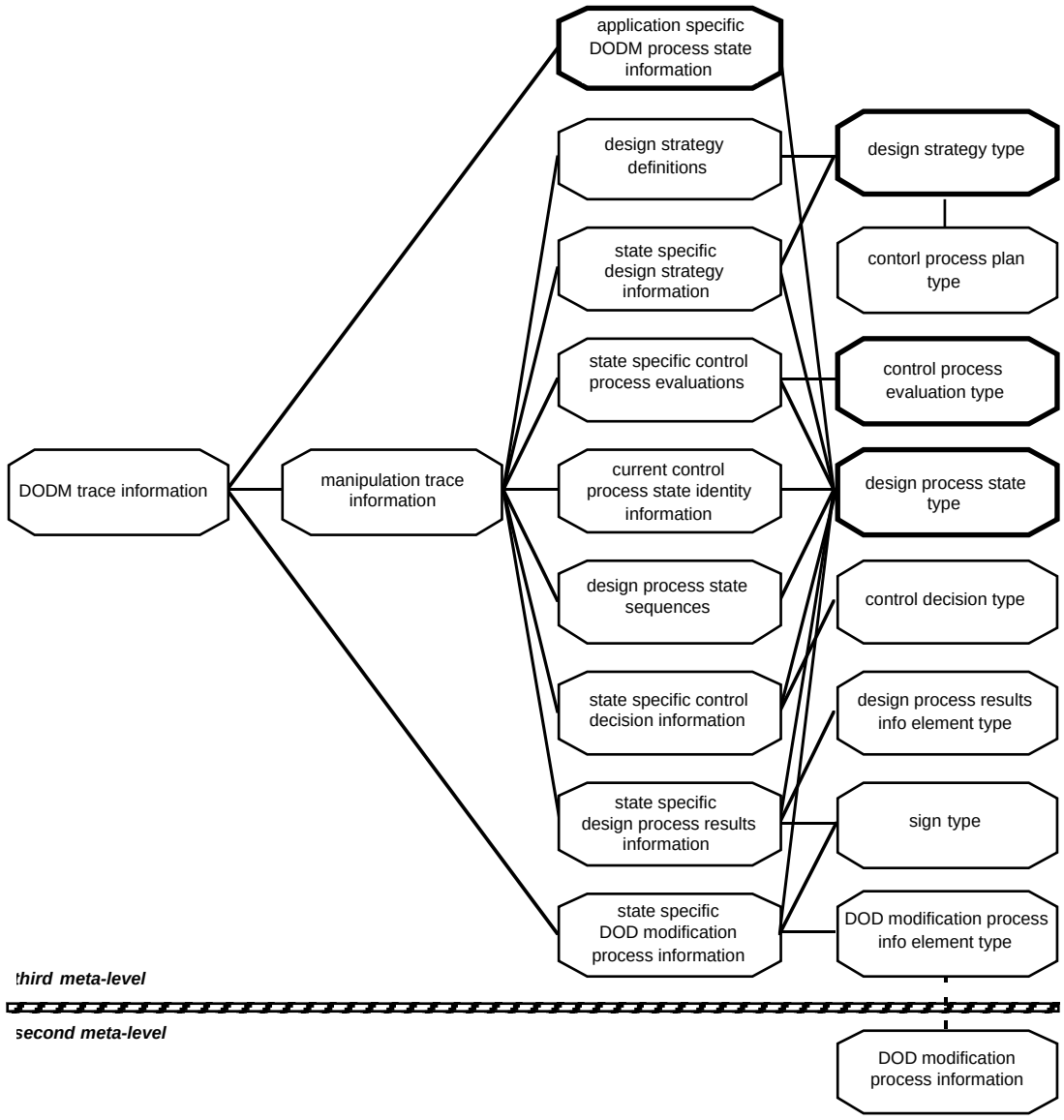


FIGURE 8.28. Composition of information type DODM trace information.

Figure 8.29 shows the structure of the information type state specific DOD modification process information, which models epistemic information about design object description modifications. It contains the relation includes-DOD-modification-process-information, which has three arguments of the sorts design-process-state, DOD-modification-process-info-element and sign, respectively. An atom includes-DOD-modification-process-information(*design-process-state1*, *DOD-modification-process-info-element1*, *sign1*) specifies that the design process state *design-process-state1* includes the modification process information *DOD-modification-process-info-element1*, with the sign *sign1* as its truth value.

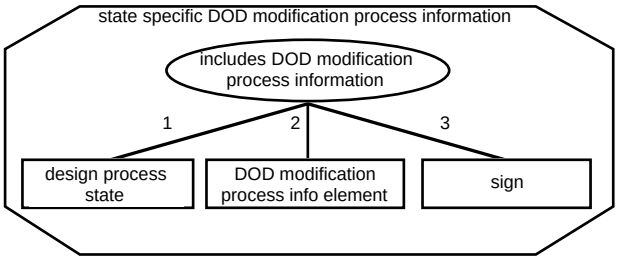


FIGURE 8.29. Structure of information type state specific DOD modification process information.

Example 8.18.
"The state of the design object description manipulation process designated State4 includes the information that the proposal is selected to replace the property that the bicycle is low priced by a property that the bicycle's price is 350 euro."

includes-DOD-modification-process-information(*State4*,
  is-selected-DOD-alteration(*AlterationOption1*), pos)
includes(DOD-modification-process-information(*State4*,
  includes-DOD-modification(*AlterationOption1*,
    deletion-of(*is-price-level*(*bicycle1*, *low*))),
  pos)
includes(DOD-modification-process-information(*State4*,
  includes-DOD-modification(*AlterationOption1*,
    addition-of(has-value(*bicycle*, *price*(*euro*), 350))),
  pos)

DODM step information. Figure 8.30 shows the composition of the information type DODM step information, which models information about the next step to be taken in a design object description manipulation process.

Refer to Chapter 6 for the composition and structure of the information type DODM process evaluations. Refer to Chapter 7 for the composition and the structure of the information type current manipulation rationale information. In the following, the other information types are explained to which the information type DODM step information refers.

Figure 8.31 shows the structure of the information type state specific DOD modification process information queries, which models queries for the information about the design object description modification process information included in a specific design process state. It contains two relations: the relation includes-which-DOD-modification-process-information has one argument of the sort design-process-state, and the relation is-included-in-which-design-process-states has two arguments of the sorts DOD-modification-process-info-element and sign, respectively.

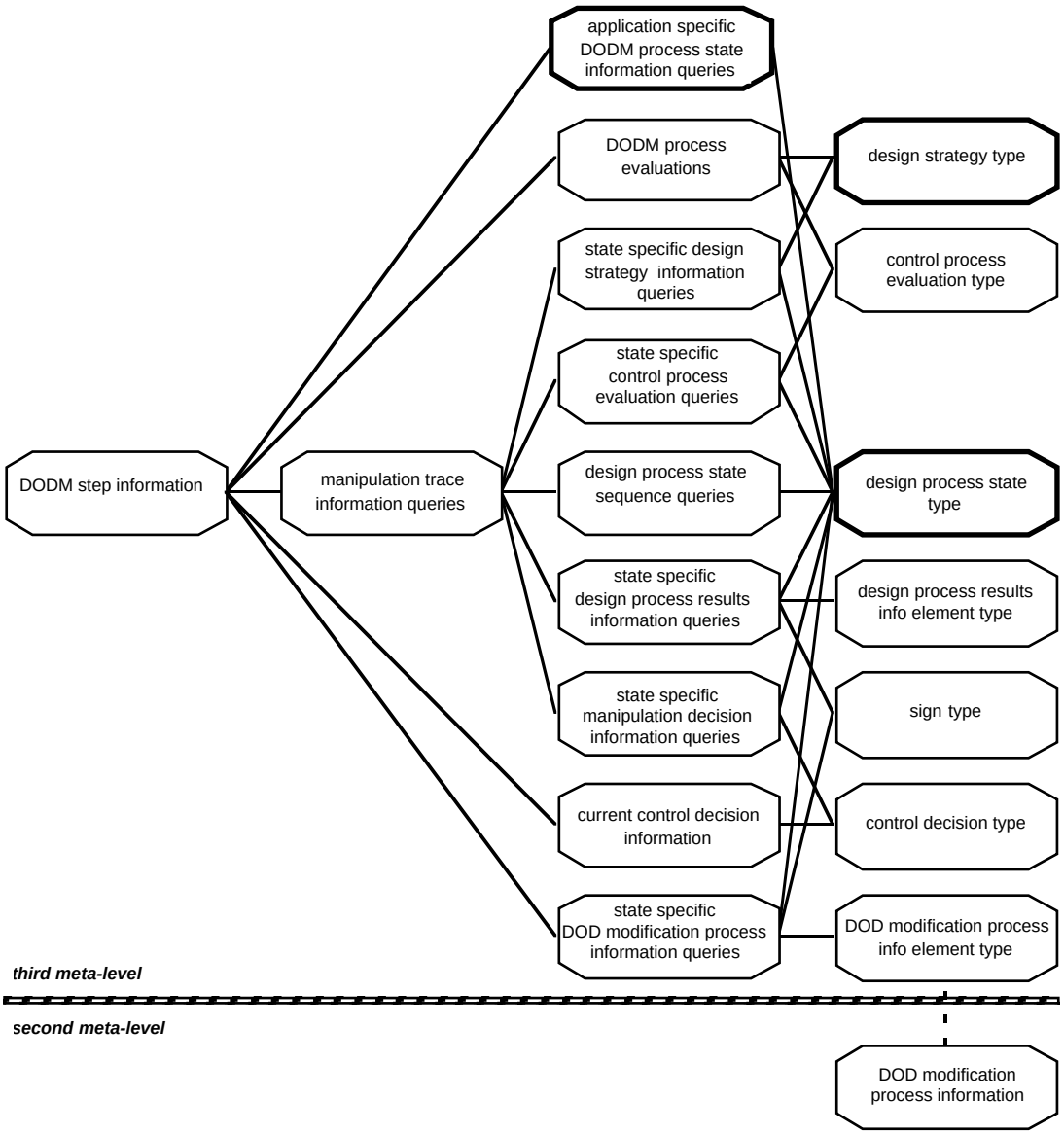


FIGURE 8.30. Composition of information type DODM step information.

An atom includes-which-DOD-modification-process-information(*design-process-state1*) specifies a query for the information about the design object description modification process included in the design process state *design-process-state1*. An atom is-included-in-which-design-process-states(*DOD-modification-process-info-element1*, *sign1*) specifies a query for the design process states that include the modification process information *DOD-modification-process-info-element1* with sign *sign1* as its truth value.

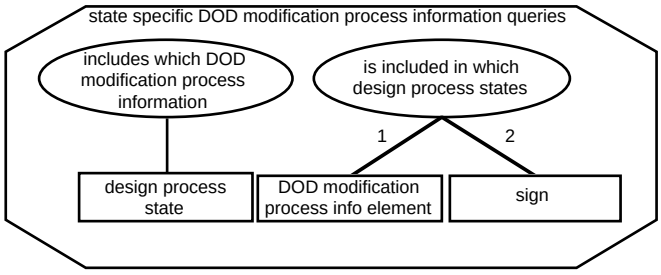


FIGURE 8.31. Structure of information type state specific DOD modification process information queries.

Example 8.19.
"There is currently a query for the modification process information included in the manipulation process state designated State4. Furthermore, there is currently a query for the design process states in which it was decided to replace the current design object description by the description named *Design Doc v0.2*."

includes-which-DOD-modification-process-information(*State4*)
is-included-in-which-design-process-states(
  is-replacement-for-current-DOD("*Design Doc v0.2*"), pos)

The information type application specific DODM process state information queries models queries for application specific information about design object description manipulation process states. For example, a relation specified within this information type can be used to model queries for information about which manipulation process states are promising.

8.3 Relation between Compositions of Process and Knowledge

The relation between the process composition and the knowledge composition of a DOD manipulation process specifies how the processes involved in DOD manipulation relate to the reflection levels for a design process.

- At the object level of a design process, the processes of current DOD maintenance and deductive DOD refinement reason with domain object information (i.e., information about domain objects and about their relations).

- At the first meta-level of a design process, the processes of DOD manipulation history maintenance and DOD modification reason with design object descriptions as well as defaults for domain object information.

- At the second meta-level of a design process, the processes of DOD manipulation history maintenance and DOD modification reason with requirement qualification sets and DOD assessments, as well as proposals for the modification of design object descriptions.

- At the third meta-level of a design process, the processes of DOD manipulation history maintenance and DOD modification reason with overall design strategies, DODM trace information and DOD manipulation process evaluations.

Figure 8.32 shows how the relation between process composition and knowledge composition of a design object description manipulation process is modelled in GDM.

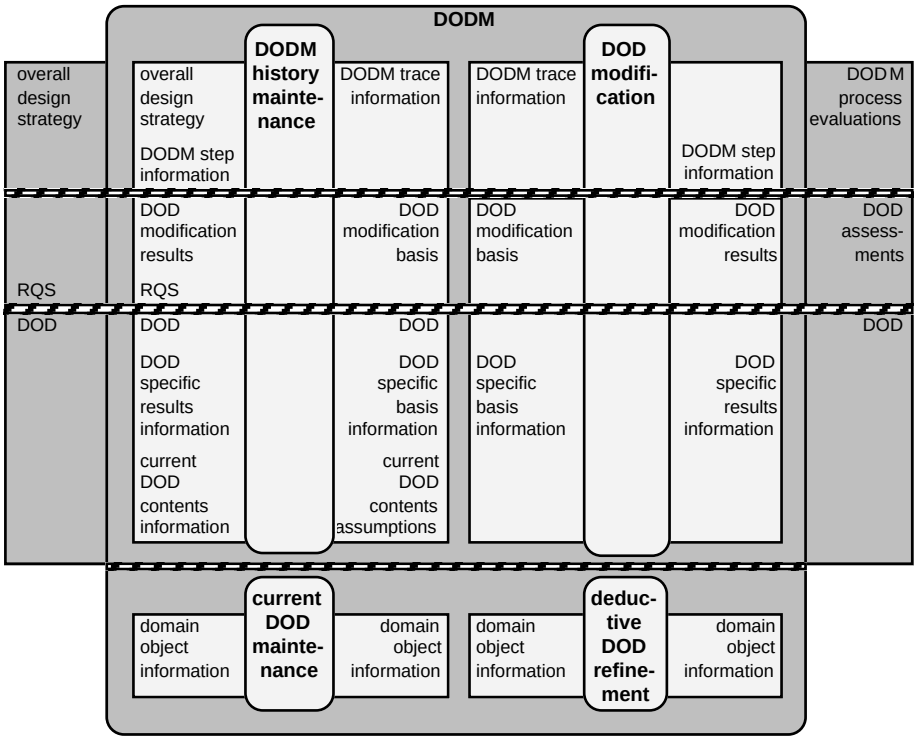


FIGURE 8.32. Relation between compositions of process and knowledge of a DOD manipulation process.

8.4 Use of the Model in Analysis and Development

This section explains how the model of a design object description manipulation process that GDM provides can be used in the analysis of practical design processes and the development of design support systems. Please note that the explanation given here should be read as an extension to Section 6.4, which focuses on the two highest levels of a design process.

Identification of design object description manipulation processes. During the analysis of applications, the following checklist can be used to determine whether or not a given application specific process is a design object description manipulation process.

- The input consists of an overall design strategy and a set of design requirements.
- The output consists of a design object description.
- The intent is to follow the given overall design strategy in order to produce a design object description that provides a solution to the given set of design requirements.

Application specific process composition. For any application specific design object description manipulation process, the sub-processes involved can be modelled by the components DODM history maintenance, DOD modification, current DOD maintenance, and deductive DOD refinement. A modeller may decide to refine the components DOD modification and DODM history maintenance to model the application specific modification of design object descriptions and maintenance of the history of design object description manipulation. Chapter 9 describes possible refinements that have proven to be useful in many practical situations.

Application specific knowledge composition. The following information types within GDM can be used to model application specific objects and relations. (Note that in the detailed textual specification of GDM in Appendix B, these information types either have names that contain application specific as a prefix, or they include objects of which the names contain Example as a prefix.)

- The information type application specific DODM process state information can be used to model application specific information about DOD manipulation process states, by means of relations on the sort design-process-state.
- The information type application specific DODM process state information queries can be used to model queries for application specific information about DOD manipulation process states, by means of relations on the sort design-process-state.
- The information type DOD alteration type can be used to model alterations to the current design object description, by means of objects of the sort DOD-alteration.
- The information type application specific DOD alteration information can be used to model application specific information about alterations to the current design object description, by means of relations on the sorts DOD-name and DOD-alteration.
- The information type application specific DOD information can be used to model application specific information about specific design object descriptions, by means of relations on the sorts DOD-name, domain-object-info-element, and sign.

The knowledge base of the component deductive DOD refinement can be used to model application specific theories about domain objects.

Chapter 9

Design Specialisations

The generic design model GDM described in Part III provides a component-based structure in terms of which design processes can be modelled. In practice, models of design processes will often have more refined structures than the three levels of process abstraction in GDM; their detailed structures are compositions of processes and knowledge that are related to specific application domains and design methods. To support the analysis of design processes, as well as the development of practical design support systems, GDM can be specialised to provide such detailed structures. This chapter presents some specialisations of GDM resulting from our research on different design applications and design themes.

Publications. *This chapter is based on earlier research on design applications, such as elevator configuration design described in Chapter 10, on design themes, such as management of design conflicts described in Chapter 14 and redesign and reuse [Brazier, Langen, Treur and Wijngaards, 1996], and on process control in dynamic environments [Brazier, Klerk, Langen and Treur, 1993].*

The generic design model GDM described in Part III provides a component-based structure in terms of which design processes can be modelled at up to three levels of process abstraction. In this way, GDM supports thorough analysis of design processes, by providing process structures and knowledge structures that can be distinguished in any design process. To support the analysis of specific types of design processes in practice, as well as the development of practical design support systems, GDM can be specialised to provide a far more detailed structure. Besides the generic support for any kind of design process, such a specialisation provides support that is specific to an application domain or a design method.

In practice, specific types of design processes will often share more refined components and knowledge structures than those modelled by GDM. The reason that these more detailed compositions have not been included in GDM is that they are (by our claim) less generic. Instead, they are compositions of processes and knowledge related to specific design methods (such as *propose-and-revise*). Though worthwhile in many cases, there are also design processes in which such detailed compositions do not apply.

For example, in many design applications the (partial) design object descriptions are at some point assessed against the given design requirements, because the design methods used in these applications cannot exclude the possibility of having generated unsatisfactory design object descriptions. But there are also processes that do not need to have an assessment subprocess to detect faulty domain object information, because by their very nature they are unable to generate unsatisfactory design object descriptions. A constraint satisfaction based process, for instance, may at some point run out of possible values for a specific variable, but those values that have been assigned to variables agree by definition with the requirements, and thus need not be assessed.

This is not to say that with GDM, the non-generic issues of design processes cannot be addressed. To support the analysis of specific classes of design processes in practice, as well as the development of practical design support systems, GDM can be specialised to provide a far more detailed structure. As explained earlier in Chapter 5, this means that the process composition and the knowledge composition of design processes as modelled by GDM can be refined to include additional processes and knowledge structures at lower levels of abstraction, which model specific design methods.

Among the sub-processes of a design process, obvious candidates for specialisation are requirement qualification set modification (RQS modification) and design object description modification (DOD modification). Both processes play a dominant role within an RQS manipulation process and a DOD manipulation process, respectively, as they determine both the steps taken and the results produced by the respective manipulation processes.

This chapter presents specialisations of RQS modification and DOD modification, which are based on the view of a modification process as a control process in a dynamic environment. This view can be explained by the fact that during most design processes, the input information of a manipulation process may be updated several times. For example, during a design process, a DOD manipulation process may receive new requirement qualification sets from an RQS manipulation process, and an RQS manipulation process may receive updates of design object description assessments from a DOD manipulation process. As described in Chapters 7 and 8, the modification process has to decide what to do with these updates and determines the next steps of the manipulation process.

This chapter is organised as follows. Section 9.1 describes a specialisation for an RQS modification process and Section 9.2 an analogous specialisation for a DOD modification process. Both specialisations are based on the view of modification as a control process in a dynamic environment. To illustrate their usability, these specialisations are reused in Chapter 10 (on elevator configuration) and Chapter 14 (on management of conflicts in design).

9.1 A Specialisation for RQS Modification

Viewing modification as a control process, the following processes can be distinguished at the two highest levels of process abstraction of an RQS modification process:

- *RQS modification* as a whole (as described in Chapter 7),
- *RQS modification analysis*, which assesses the current requirement qualification set and which evaluates the current state of the requirement qualification set manipulation process, and
- *RQS modification determination*, which determines new modifications of the current requirement qualification set or any other course of action to be taken (i.e., termination of the requirement qualification set manipulation process, replacement of the current requirement qualification set by an earlier generated requirement qualification set, deductive refinement of the current requirement qualification set, or query and retrieval of the requirement qualification set manipulation history).

Figure 9.1 shows the types of information that the sub-processes of an RQS modification process use as input and the types of information that these processes produce as output. Note that these types of information have already been introduced in Chapter 7.

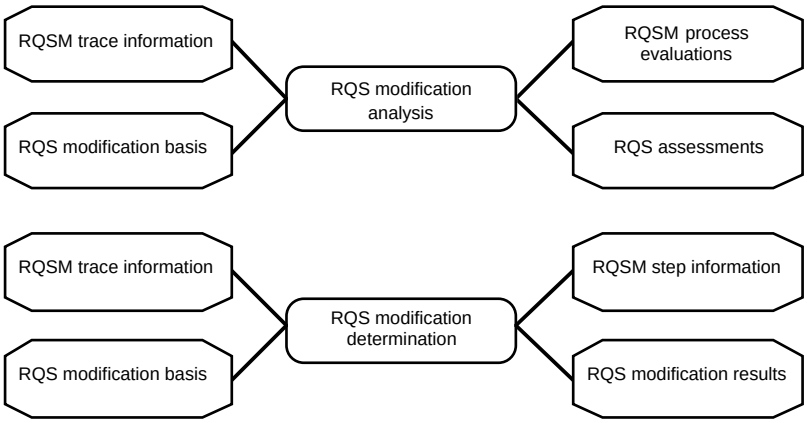


FIGURE 9.1. Input and output of sub-processes of RQS modification.

Figure 9.2 shows the process abstraction relations between an RQS modification process and its sub-processes.

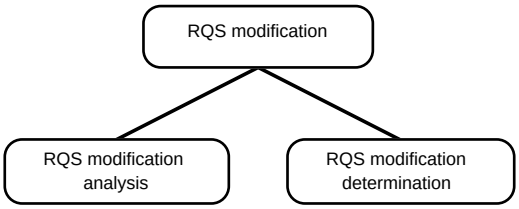


FIGURE 9.2. Two levels of abstraction for an RQS modification process.

Figure 9.3 shows the possibilities for information exchange between processes involved in RQS modification, modelled as information links within the component RQS modification.

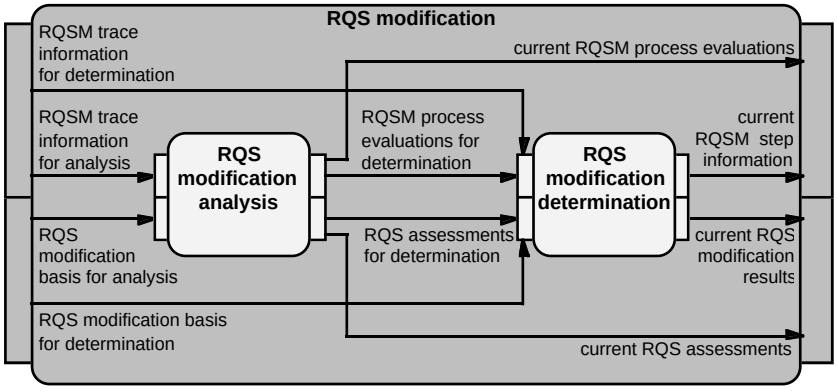


FIGURE 9.3. Information exchange between processes involved in RQS modification.

Table 9.1 presents a possible model of the task control for an RQS modification process, given its sub-processes and possibilities for information exchange.

TABLE 9.1. Task control for an RQS modification process.

State	*Information link(s) updated next*	*Component(s) activated next*
RQS modification has started.	RQSM trace information for analysis, RQSM trace information for determination, RQS modification basis for analysis, and RQS modification basis for determination.	RQS modification analysis.
RQS modification analysis has terminated.	RQSM process evaluations for determination, and RQS assessments for determination.	RQS modification determination.
RQS modification determination has terminated.	Current RQSM process evaluations, current RQSM step information, current RQS assessments, and current RQS modification results.	None (RQS modification terminates).

9.1.1 A specialisation for RQS modification analysis

The following processes are involved at the two highest levels of process abstraction of an RQS modification analysis process:

- *RQS modification analysis* as a whole,
- *RQS assessment*, which assesses the current requirement qualification set (with respect to the satisfaction of the design requirements it includes as well as its fulfilment),
- *RQS modification evaluation*, which evaluates the effects of the most recent set of modifications that led to the current requirement qualification set,
- *RQSM process evaluation*, which evaluates the requirement qualification set manipulation process (up to its current state) against the current overall design strategy.

Figure 9.4 shows the types of information that the sub-processes of an RQS modification analysis process use as input and the types of information that these processes produce as output. Note that these information types have already been introduced in Chapter 7, with the exception of RQS modification evaluations and epistemic RQS modification evaluation information.

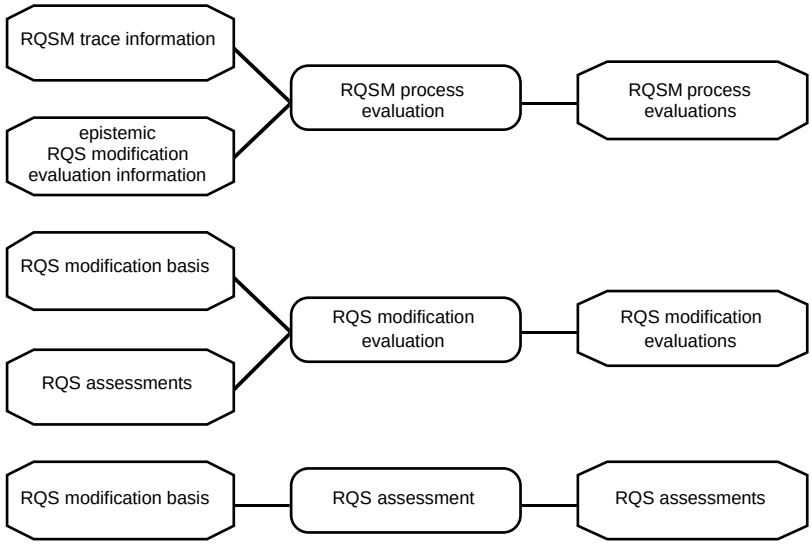


FIGURE 9.4. Input and output of sub-processes of RQS modification analysis.

Figure 9.5 shows the composition of the information type RQS modification evaluations, which models evaluations of alterations (i.e., single modifications or compositions of modifications) to requirement qualification sets. Note that the information types RQS name type and RQS alteration type have already been introduced in Chapter 6 and Chapter 7, respectively.

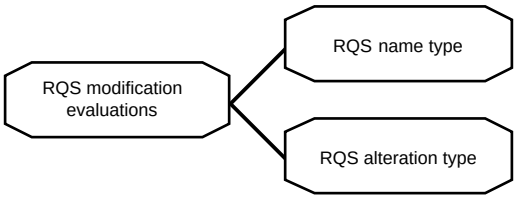


FIGURE 9.5. Composition of information type RQS modification evaluations.

Figure 9.6 shows the structure of the information type RQS modification evaluations. It contains the relation is-acceptable-RQS-alteration-to, which has two arguments of the sort RQS-alteration and RQS-name, respectively. An atom is-acceptable-RQS-alteration-to(*RQS-alteration1*, *RQS-name1*) specifies that application of the alteration *RQS-alteration1* to the requirement qualification set *RQS-name1* is acceptable.

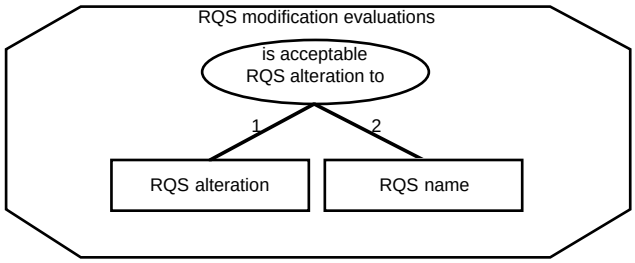


FIGURE 9.6. Structure of information type RQS modification evaluations.

Example 9.1.
"The modification to the set RQS1, involving the addition of a new requirement that the frame of the bicycle should contain shock absorbers, is acceptable."

is-acceptable-RQS-alteration-to(
addition-of(is-defined-as(*R1a*, contains(*frame1*, *shock-absorbers*))), RQS1)

Figure 9.7 shows the composition of the information type epistemic RQS modification evaluation information, which models epistemic information about evaluations of requirement qualification set modifications.

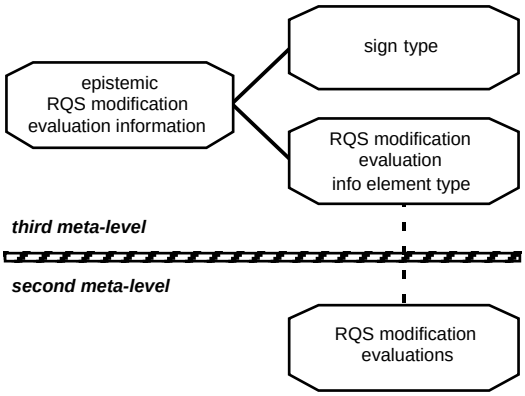


FIGURE 9.7. Composition of information type epistemic RQS modification evaluation information.

Figure 9.8 shows the structure of the information type epistemic RQS modification evaluation information. It contains the relation holds, which has two arguments of the sorts RQS-modification-evaluation-info-element and sign, respectively. An atom holds(*RQS-modification-evaluation-info-element1*, *sign1*) specifies that the current evaluations of requirement qualification set modifications include the information *RQS-modification-evaluation-info-element1*, with sign *sign1* as its truth value. If *sign1* equals pos, it means that *RQS-modification-evaluation-info-element1* is known to hold (i.e., is true); if *sign1* equals neg, it means that *RQS-modification-evaluation-info-element1* is known to not hold (i.e., is false); if *sign1* equals unk, it means that it is unknown whether or not *RQS-modification-evaluation-info-element1* holds.

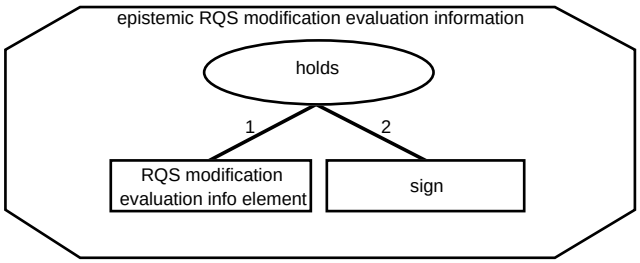


FIGURE 9.8. Structure of information type epistemic RQS modification evaluation information.

Example 9.2.
"It is known that the last alteration to the requirement qualification set named RQS1, which involved the addition of a new requirement that the frame of the bicycle should contain shock absorbers, is acceptable."

holds(is-acceptable-RQS-alteration-to(
addition-of(is-defined-as(*R1a*, contains(*frame1*, *shock-absorbers*))), RQS1), pos)

Figure 9.9 shows the abstraction relations between an RQS modification analysis process and its sub-processes.

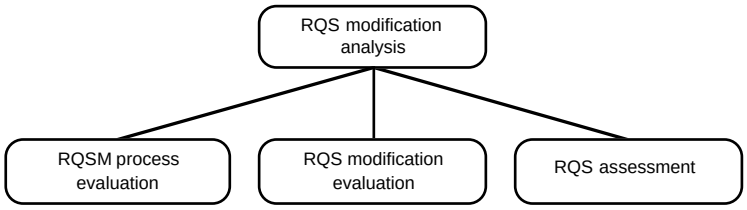


FIGURE 9.9. Two levels of abstraction for an RQS modification analysis process.

Figure 9.10 shows the possibilities for information exchange between processes involved in RQS modification analysis, modelled as information links within the component RQS modification analysis.

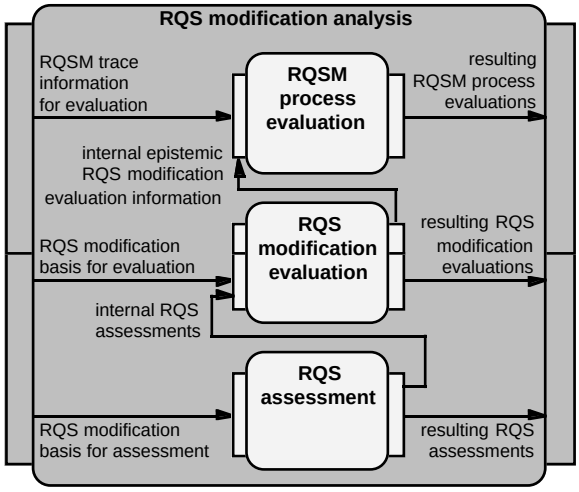


FIGURE 9.10. Information exchange between processes involved in RQS modification analysis.

Table 9.2 presents a possible model of the task control for an RQS modification analysis process, given its sub-processes and possibilities for information exchange. (Note that sequential task control is assumed, but parallel control is equally justifiable.)

TABLE 9.2. Task control for an RQS modification analysis process.

State	*Information link(s) updated next*	*Component(s) activated next*
RQS modification analysis has started.	RQSM trace information for evaluation, RQS modification basis for evaluation, and RQS modification basis for assessment.	RQS assessment.
RQS assessment has terminated.	Internal RQS assessments.	RQS modification evaluation.
RQS modification evaluation has terminated.	Internal epistemic RQS modification evaluation information.	RQSM process evaluation.
RQSM process evaluation has terminated.	Resulting RQSM process evaluations, resulting RQS modification evaluations, and resulting RQS assessments.	None (RQS modification analysis terminates).

9.1.1.1 A specialisation for RQS assessment

The following processes are involved at the two highest levels of process abstraction of a requirement qualification set assessment process:

- *RQS assessment* as a whole, which assesses the current requirement qualification set,
- *design requirement assessment preparation*, which sets targets and assumptions for the assessment of the design requirements included in the current requirement qualification set,
- *design requirement assessment*, which assesses the design requirements designated by given targets,
- *design requirement assessment completion*, which establishes a relation between the current requirement qualification set and given epistemic information about assessments of design requirement, and
- *overall RQS assessment*, which assesses the current requirement qualification set on the basis of epistemic information about assessments of the design requirements that this set includes.

Figure 9.11 shows the types of information that the sub-processes of an RQS assessment process use as input and the types of information that these processes produce as output. Note that most of these information types have already been introduced in Chapter 7; in the following, the remaining information types are explained.

Figure 9.12 shows the structure of the information type basic evaluation targets, which models targets for assessment of the design requirements included in the current requirement qualification set and (in the context of DOD assessment, to be explained later) for assessment of a design object description against these design requirements. It contains the relation is-to-be-determined, which has two arguments of the sorts basic-evaluation-info-element and sign, respectively.

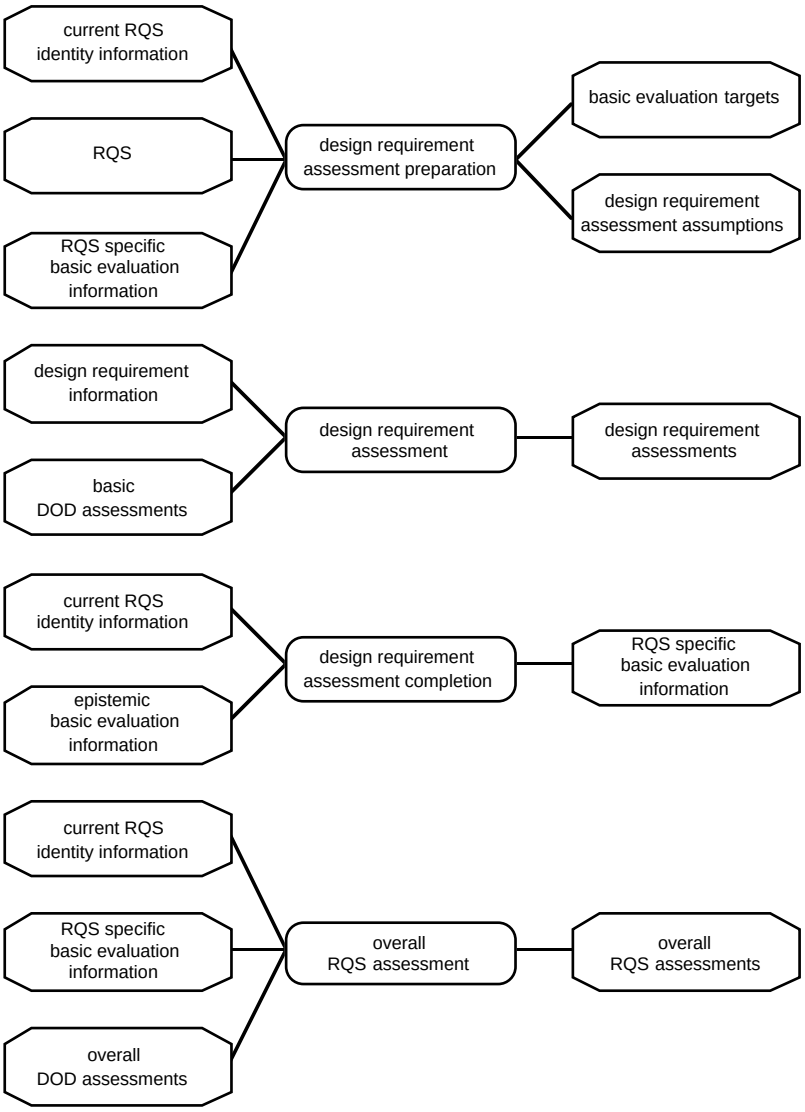


FIGURE 9.11. Input and output of sub-processes of RQS assessment.

An atom is-to-be-determined(*basic-evaluation-info-element1*, *sign1*) specifies that a current target for design requirement assessment is the information *basic-evaluation-info-element1*, with sign *sign1* as its target value. If *sign1* equals pos, it means that *basic-evaluation-info-element1* has to be confirmed; if *sign1* equals neg, it means that *basic-evaluation-info-element1* has to be rejected.

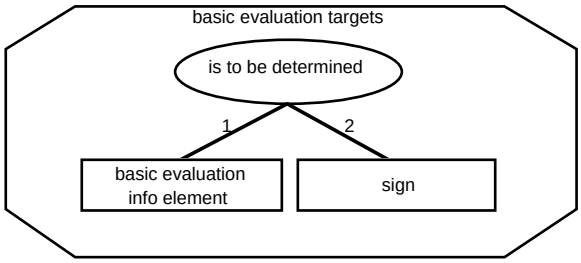


FIGURE 9.12. Structure of information type basic evaluation targets.

Example 9.3.
"One fact to be confirmed is that the requirement named R1 can be satisfied by some design object description, and another fact to be rejected is that the qualified requirement named QR2 can be satisfied by some design object description."

is-to-be-determined(can-be-satisfied(*R1*), pos)
is-to-be-determined(can-be-satisfied(*QR2*), neg)

Figure 9.13 shows the composition of the information type design requirement assessment assumptions, which models assumptions relevant for the assessment of design requirements included in the current requirement qualification set.

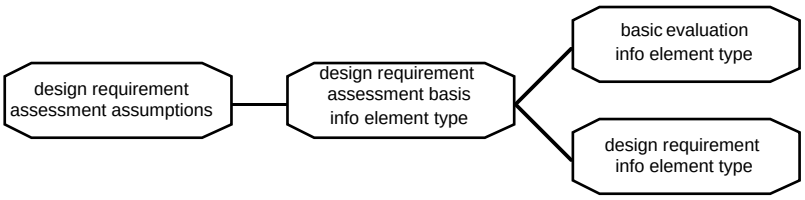


FIGURE 9.13. Composition of information type design requirement assessment assumptions.

The information type design requirement assessment basis info element type models elements of information used as a basis for a design requirement assessment process: definitions of the design requirements included in the current requirement qualification set, and basic evaluations that have already been determined for the current requirement qualification set. Within this information type, the sorts basic-evaluation-info-element and design-requirement-info-element (described in Chapter 6) are defined as sub-sorts of the sort design-requirement-assessment-basis-info-element.

Figure 9.14 shows the structure of the information type design requirement assessment assumptions. It contains the relation is-to-be-assumed, which has two arguments of the sorts design-requirement-assessment-basis-info-element and sign, respectively. An atom is-to-be-

assumed(*design-requirement-assessment-basis-info-element1*, *sign1*) specifies that the information *design-requirement-assessment-basis-info-element1*, with sign *sign1* as its assumed value, is a current assumption for design requirement assessment. If *sign1* equals pos, it means that *design-requirement-assessment-basis-info-element1* is assumed to be true; if *sign1* equals neg, it means that *design-requirement-assessment-basis-info-element1* is assumed to be false.

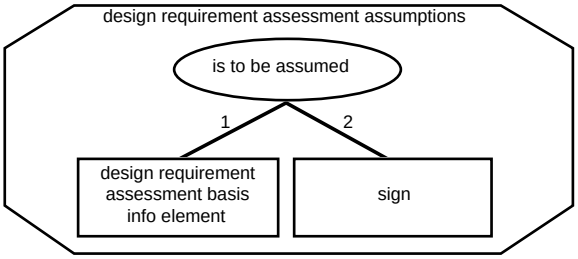


FIGURE 9.14. Structure of information type design requirement assessment assumptions.

Example 9.4.
"One assumption is that the requirement named R1 states that the bicycle to be designed must be suited for riding in mountains (which is the information needed for the assessment of R1). Another assumption is that the design object description named DOD1 satisfies the requirement named R1."

is-to-be-assumed(is-defined-as(*R1*, *is-suitable-for-terrain-type*(*bicycle1*, *mountains*)), pos)
is-to-be-assumed(satisfies(*DOD1*, *R1*), pos)

Figure 9.15 shows the structure of the information type epistemic basic evaluation information, which models epistemic information about design requirement assessments and basic DOD assessments. It contains the relation holds, which has two arguments of the sorts basic-evaluation-info-element and sign, respectively. An atom holds(*basic-evaluation-info-element1*, *sign1*) specifies that the basic evaluation information *basic-evaluation-info-element1* holds, with sign *sign1* as its truth value. If *sign1* equals pos, it means that *basic-evaluation-info-element1* is known to hold (i.e., is true); if *sign1* equals neg, it means that *basic-evaluation-info-element1* is known to not hold (i.e., is false)); if *sign1* equals unk, it means that it is unknown whether or not *basic-evaluation-info-element1* holds.

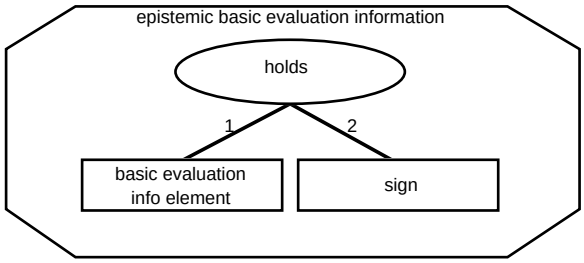


FIGURE 9.15. Structure of information type epistemic basic evaluation information.

Example 9.5.
"It is known to be true that the requirement named R1 can be satisfied and it is not known whether or not the qualified requirement named QR2 can be satisfied."

holds(can-be-satisfied(*R1*), pos)
holds(can-be-satisfied(*QR2*), unk)

Figure 9.16 shows the abstraction relations between an RQS assessment process and its sub-processes.

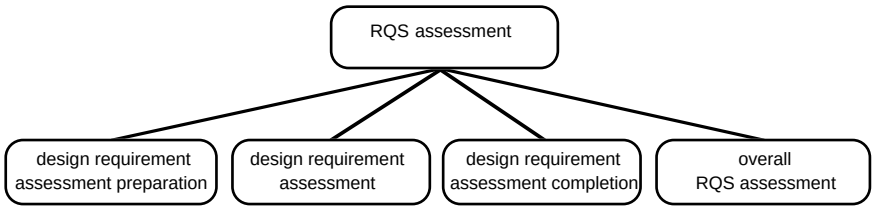


FIGURE 9.16. Two levels of abstraction for an RQS assessment process.

Figure 9.17 shows the possibilities for information exchange between processes involved in RQS assessment, modelled as information links within the component RQS assessment. Note that RQS assessment is positioned at the second meta-level of a design process.

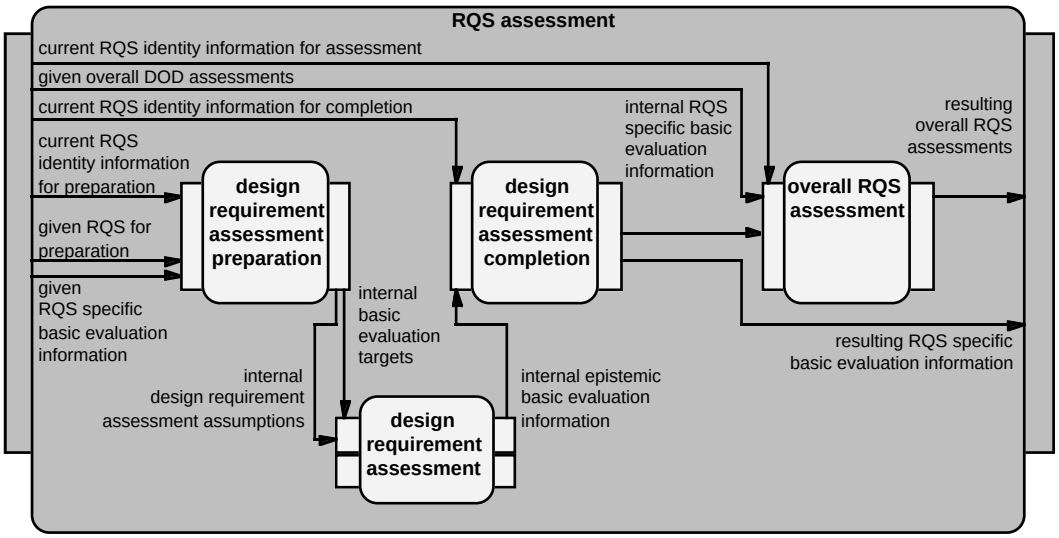


FIGURE 9.17. Information exchange between processes involved in RQS assessment.

Table 9.3 presents a possible model of the task control for an RQS assessment process, given its sub-processes and possibilities for information exchange.

TABLE 9.3. Task control for an RQS assessment process.

State	*Information link(s) updated next*	*Component(s) activated next*
RQS assessment has started.	Current RQS identity information for preparation, given RQS for preparation, and given RQS specific basic evaluation information.	Design requirement assessment preparation.
Design requirement assessment preparation has terminated.	Internal basic evaluation targets, and internal design requirement assessment assumptions.	Design requirement assessment.
Design requirement assessment has terminated.	Current RQS identity information for completion, and internal epistemic basic evaluation information.	Design requirement assessment completion.
Design requirement assessment completion has terminated.	Current RQS identity information for assessment, given overall DOD assessments, and internal RQS specific basic evaluation information.	Overall RQS assessment.
Overall RQS assessment has terminated.	Resulting overall RQS assessments, and resulting RQS specific basic evaluation information.	None (RQS assessment terminates).

The remainder of this section describes the knowledge used by the sub-processes of an RQS assessment process. (For the sake of brevity, the description of three sub-processes is not specified in detail.)

Knowledge for design requirement assessment preparation. Preparation for design requirement assessment involves the knowledge that each individual design requirement included in the current requirement qualification set is to be assessed. Preparation also involves the knowledge that all design requirements included in the current requirement qualification set, as well as all basic evaluation information that is already available for the current requirement qualification set, have to be assumed to hold for design requirement assessment. Using DESIRE (explained in Chapter 5), this knowledge is modelled by the following rules within the knowledge base of the component design requirement assessment preparation:

```
if is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(R: requirement-name, E: domain-object-info-expression), pos)
then is-to-be-determined(can-be-satisfied(R: requirement-name), pos);

if is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(R: requirement-name, E: domain-object-info-expression), pos)
then is-to-be-assumed(
  is-defined-as(R: requirement-name, E: domain-object-info-expression), pos);

if is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
then is-to-be-determined(can-be-satisfied(QR: qualified-requirement-name), pos);

if is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
then is-to-be-assumed(
  is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos);

if is-current-RQS(CurRN: RQS-name)
  and includes-basic-evaluation-information(
    CurRN: RQS-name, E: basic-evaluation-info-element, S: sign)
then is-to-be-assumed(E: basic-evaluation-info-element, S: sign);
```

Knowledge for design requirement assessment. Design requirement assessment involves knowledge to determine whether or not a specific design requirement can be satisfied. This knowledge (used within the component design requirement assessment) makes use of the definitions of design requirements (as described in Chapter 6) and reads as follows.

A specific design requirement can be satisfied if there already exists a design object description that satisfies the requirement. A specific requirement cannot be satisfied if there is a logical inconsistency within the domain object information expression defined for this requirement. A specific qualified requirement cannot be satisfied if it is not possible to satisfy the right combination of requirements from the requirement list defined for this qualified requirement. (Whether a combination is 'right' is determined by the qualification defined for this qualified requirement. For example, the qualification 'any' demands that at least one of the requirements in the list is satisfied by a design object description.)

Knowledge for design requirement assessment completion. Completion of design requirement assessment involves the knowledge that for each element of basic evaluation information that currently holds, a relation has to be established to the current requirement qualification set. This knowledge is modelled by the following rule within the knowledge base of the component design requirement assessment completion:

```
if is-current-RQS(CurRN: RQS-name)
  and holds(E: basic-evaluation-info-element, S: sign)
then includes-basic-evaluation-information(
  CurRN: RQS-name, E: basic-evaluation-info-element, S: sign);
```

Knowledge for overall RQS assessment. Overall RQS assessment involves knowledge (used within the component overall RQS assessment) to determine whether or not a specific requirement qualification set can be fulfilled by some design object description. A requirement qualification set can be fulfilled if there already exists a design object description that fulfils the requirement qualification set. A specific requirement qualification set cannot be fulfilled if it includes a qualified requirement that cannot be satisfied.

9.1.1.2 RQS modification evaluation

The task of an RQS modification evaluation process is to evaluate RQS alterations. This evaluation may involve the results of assessing the current requirement qualification set, in order to determine the acceptability of a specific requirement qualification set alteration. For example, it may be that the last alteration is accepted only if it has succeeded to remove a design requirement that cannot be satisfied.

9.1.1.3 RQSM process evaluation

The task of an RQSM process evaluation process is to evaluate the current requirement qualification set manipulation process on the basis of a specific overall design strategy. This evaluation may be performed using generic, domain-specific or heuristic knowledge. Using generic knowledge requires overall design strategies to be formulated in terms of non-subjective criteria. Using domain-specific knowledge or heuristic knowledge (e.g., stating the maximum number of manipulation attempts to be tried) requires a thorough understanding of how requirement qualification set manipulation processes take place in specific application domains.

9.1.2 A specialisation for RQS modification determination

This section describes a specialisation for RQS modification determination, which is the second of two sub-processes of an RQS modification process. The specialisation is based on the "object-act" paradigm of first selecting an object and then determining an action to be performed on that object. (The "object-act" paradigm is the basis of object oriented modelling, design and development.) According to this specialisation, the following processes are involved at the two highest levels of process abstraction of RQS modification determination:

- *RQS modification determination* as a whole,
- *RQS modification focus determination*, which determines the part of the current requirement qualification set for which a modification is to be determined next,
- *RQS modification method determination*, which determines the method by means of which a modification to the current focus is to be determined next,
- *RQS modification method execution*, which executes the current method in order to determine a modification to the current focus.

Figure 9.18 shows the types of information that the sub-processes of an RQS modification determination process use as input and the types of information that these processes produce as output. Note that these information types have already been introduced in Chapter 7, with the exception of RQS modification focus and current modification method identity information.

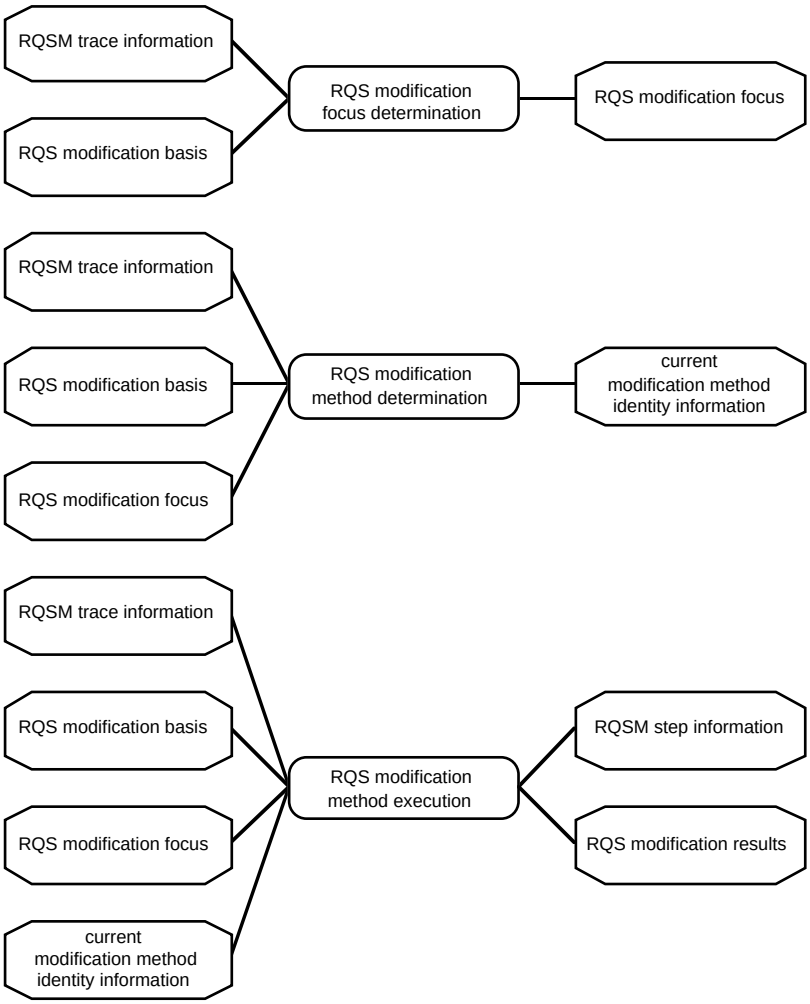


FIGURE 9.18. Input and output of sub-processes of RQS modification determination.

Figure 9.19 shows the structure of the information type RQS modification focus, which models information about which part (i.e., a sub-set of design requirements) of an requirement qualification set is currently in focus. It contains the relation is-in-current-RQS-modification-focus, that has one argument of the sort design-requirement-info-element. An atom is-in-current-RQS-modification-focus(*design-requirement-info-element1*) specifies that one of the design requirements from the current requirement qualification set that is a current subject of modification is the one designated by the information element *design-requirement-info-element1*.

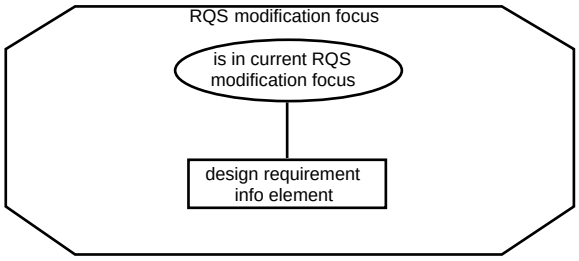


FIGURE 9.19. Structure of information type RQS modification focus.

Example 9.6.
"Current subjects of requirement qualification set modification are the requirement named R2 (which states that the price of the bicycle should be low) and the qualified requirement named QR2 (which states that requirements R3 and R2 should both be satisfied, if possible, and if that is not possible, then satisfying R3 is preferred over satisfying R2, if possible)."

is-in-current-RQS-modification-focus(is-defined-as(*R2*, *is-price-level*(*bicycle1*, *low*)))
is-in-current-RQS-modification-focus(is-defined-as(*QR2*, all-possible, [*R3*, *R2*]))

Figure 9.20 shows the structure of the information type current modification method identity information, which models information about which modification method currently applies. It contains the relation is-current-modification-method, with one argument of the sort modification-method. An atom is-current-modification-method(*modfication-method1*) specifies that the current modification method is named *modification-method1*.

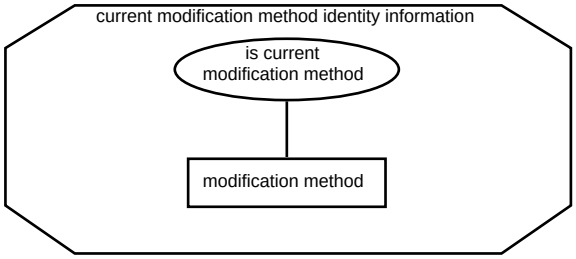


FIGURE 9.20. Structure of information type current modification method identity information.

Example 9.7.
"The current method for modification is hierarchical decomposition, which means that the design requirements in focus are to be refined in order to produce more detailed design requirements."

is-current-modification-method(*hierarchical-decomposition*)

Figure 9.21 shows the abstraction relations between an RQS modification determination process and its sub-processes.

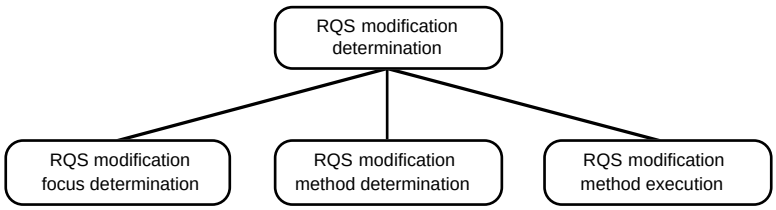


FIGURE 9.21. Two levels of abstraction for an RQS modification determination process.

Figure 9.22 shows the possibilities for information exchange between processes involved in RQS modification determination, modelled as information links within the component RQS modification determination. Note that RQS modification determination is positioned at the second and third meta-level of a design process.

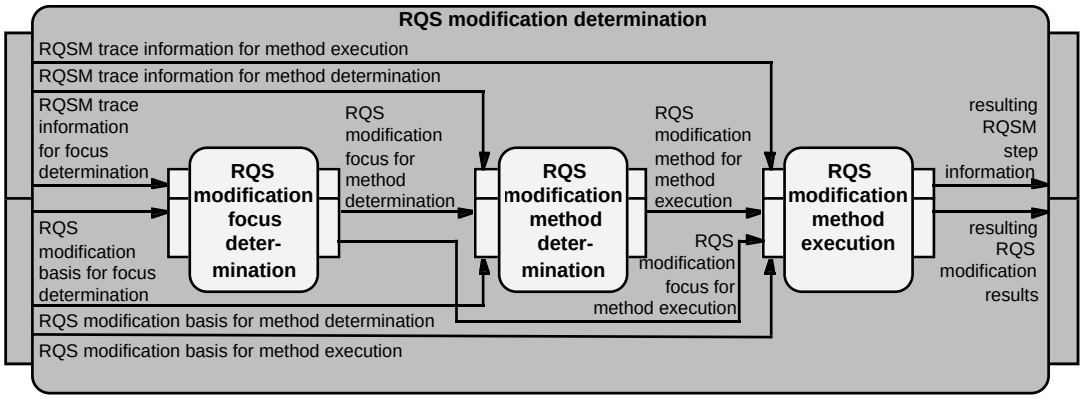


FIGURE 9.22. Information exchange between processes involved in RQS modification determination.

Table 9.4 presents a possible model of the task control for an RQS modification determination process, given its sub-processes and possibilities for information exchange.

TABLE 9.4. Task control for an RQS modification determination process.

State	*Information link(s) updated next*	*Component(s) activated next*
RQS modification determination has started.	RQSM trace information for focus determination, and RQS modification basis for focus determination.	RQS modification focus determination.
RQS modification focus determination has terminated.	RQS modification focus for method determination, RQSM trace information for method determination, and RQS modification basis for method determination.	RQS modification method determination.
RQS modification method determination has terminated.	RQS modification focus for method execution, RQS modification method for method execution, RQSM trace information for method execution, and RQS modification basis for method execution.	RQS modification method execution.
RQS modification method execution has terminated.	Resulting RQSM step information, and resulting RQS modification results.	None (RQS modification determination terminates).

The knowledge used by the three sub-processes of an RQS modification determination process can be expected to be of an application specific or heuristic nature. Typically, these sub-processes will use knowledge about a specific method for the transformation, translation, hierarchical decomposition, reduction or extension of requirement qualification sets. (This non-exhaustive list of possibilities to determine a modification of a requirement qualification set has been mentioned by Treur [Treur, 1991]).

9.2 A Specialisation for DOD Modification

Viewing modification as a control process, the following processes can be distinguished at the two highest levels of process abstraction of a DOD modification process:

- *DOD modification* as a whole (as described in Chapter 8),
- *DOD modification analysis*, which assesses the current design object description and which evaluates the current state of the design object description manipulation process, and
- *DOD modification determination*, which determines new modifications of the current design object description or any other course of action to be taken (i.e., termination of the design object description manipulation process, replacement of the current design object description by an earlier generated design object description, deductive refinement of the current design object description, or query and retrieval of the design object description manipulation history).

Figure 9.23 shows the types of information that the sub-processes of a DOD modification process use as input and the types of information that these processes produce as output. Note that these types of information have already been introduced in Chapter 8.

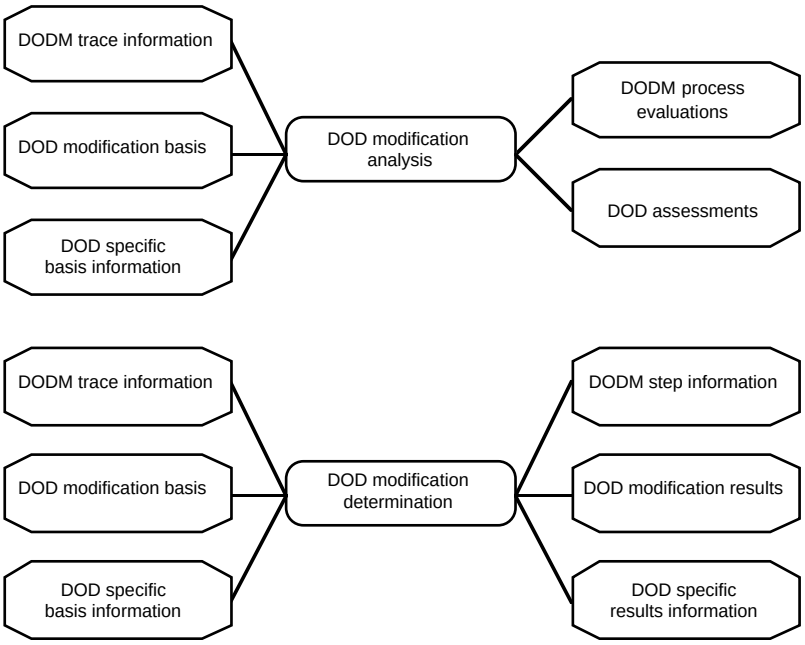


FIGURE 9.23. Input and output of sub-processes of DOD modification.

Figure 9.24 shows the abstraction relations between a DOD modification process and its sub-processes.

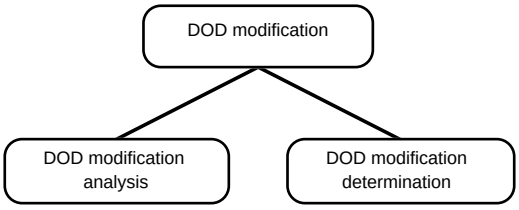


FIGURE 9.24. Two levels of abstraction for a DOD modification process.

Figure 9.25 shows the possibilities for information exchange between processes involved in DOD modification, modelled as information links within the component DOD modification.

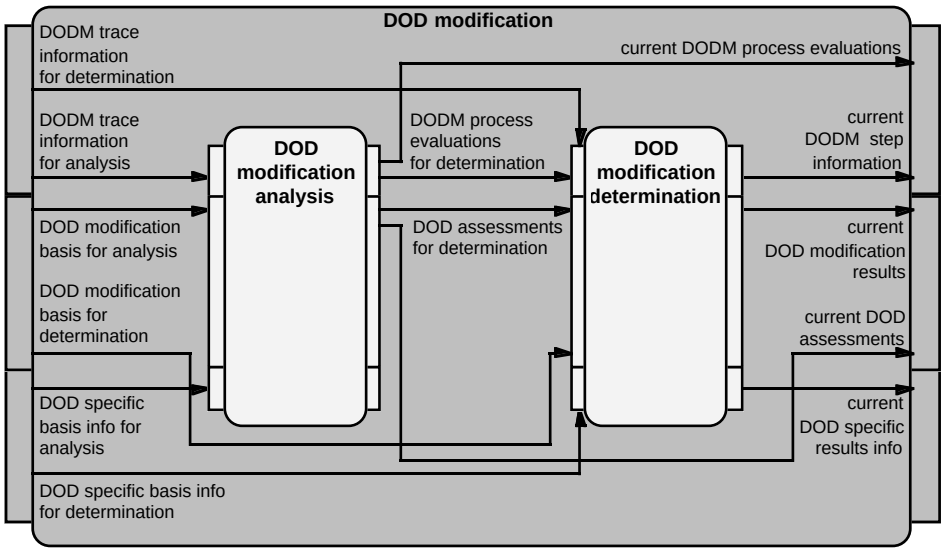


FIGURE 9.25. Information exchange between processes involved in DOD modification.

Table 9.5 presents a possible model of the task control for a DOD modification process, given its sub-processes and possibilities for information exchange.

TABLE 9.5. Task control for a DOD modification process.

State	*Information link(s) updated next*	*Component(s) activated next*
DOD modification has started.	DODM trace information for analysis, DODM trace information for determination, DOD modification basis for analysis, DOD specific basis info for analysis, DOD modification basis for determination, and DOD specific basis info for determination.	DOD modification analysis.
DOD modification analysis has terminated.	DODM process evaluations for determination, and DOD assessments for determination.	DOD modification determination.
DOD modification determination has terminated.	Current DODM process evaluations, current DODM step information, current DOD assessments, current DOD modification results, and current DOD specific results info.	None (DOD modification terminates).

9.2.1 A specialisation for DOD modification analysis

Analogously to RQS modification analysis, the following processes are involved at the two highest levels of process abstraction of a DOD modification analysis process:

- *DOD modification analysis* as a whole,
- *DOD assessment*, which assesses the current design object description,
- *DOD modification evaluation*, which evaluates the effects of the most recent set of modifications that led to the current design object description,
- *DODM process evaluation*, which evaluates the design object description manipulation process (up to its current state) against the current overall design strategy.

Figure 9.26 shows the types of information that the sub-processes of a DOD modification analysis process use as input and the types of information that these processes produce as output. Note that these information types have already been introduced in Chapter 8, with the exception of DOD modification evaluations and epistemic DOD modification evaluation information.

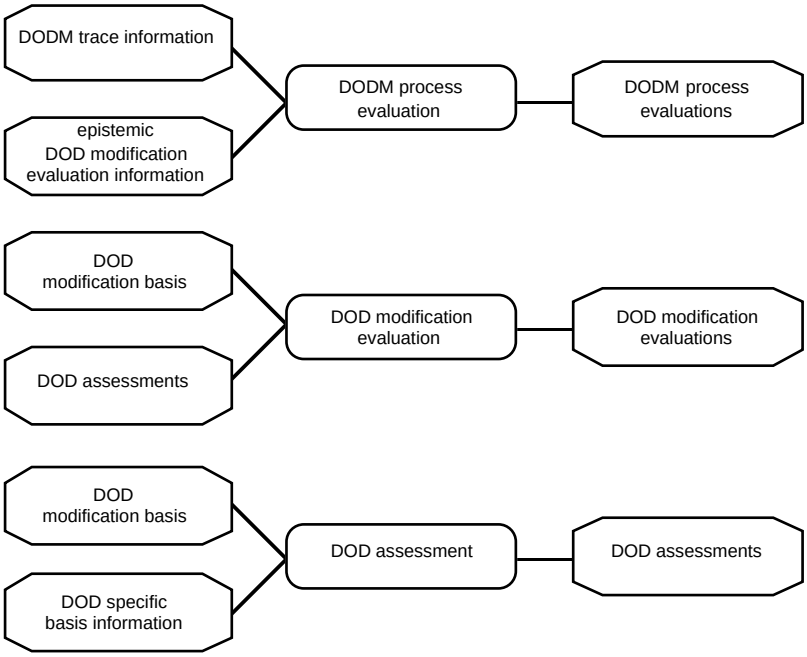


FIGURE 9.26. Input and output of sub-processes of DOD modification analysis.

Figure 9.27 shows the composition of the information type DOD modification evaluations, which models evaluations of alterations (i.e., single modifications or compositions of modifications) to design object descriptions. Note that the information types DOD name type and DOD alteration type have already been introduced in Chapter 6 and Chapter 8, respectively.

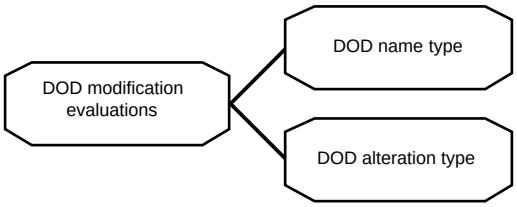


FIGURE 9.27. Composition of information type DOD modification evaluations.

Figure 9.28 shows the structure of the information type DOD modification evaluations. It contains the relation is-acceptable-DOD-alteration-to, which has two arguments of the sort DOD-alteration and DOD-name, respectively. An atom is-acceptable-DOD-alteration-to(*DOD-alteration1*, *DOD-name1*) specifies that application of the alteration *DOD-alteration1* to the design object description *DOD-name1* is acceptable.

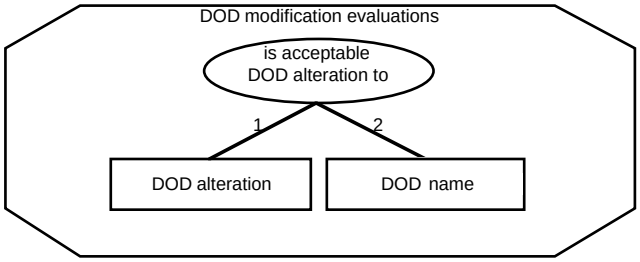


FIGURE 9.28. Structure of information type DOD modification evaluations.

Example 9.8.
"The modification to design object description DOD1, involving the deletion of an unsatisfactory part from the current design object description, is acceptable."

is-acceptable-DOD-alteration-to(
  deletion-of(domain-object-information(is-part-of(*frame1*, *bicycle1*), pos)), DOD1)
is-acceptable-DOD-alteration-to(
  deletion-of(domain-object-information(contains(*frame1*, shock-absorbers), pos)), DOD1)

Figure 9.29 shows the composition of the information type epistemic DOD modification evaluation information, which models epistemic information about evaluations of design object descriptions.

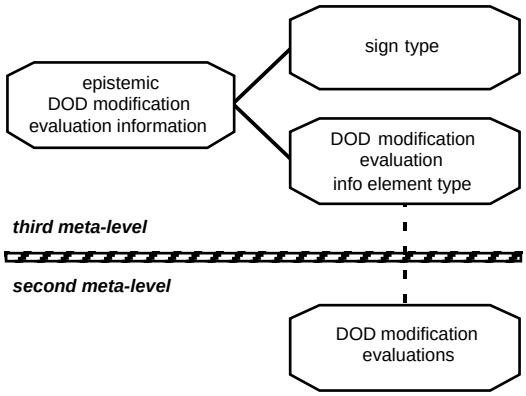


FIGURE 9.29. Composition of information type epistemic DOD modification evaluation information.

Figure 9.30 shows the structure of the information type epistemic DOD modification evaluation information. It contains the relation holds, which has two arguments of the sorts DOD-modification-evaluation-info-element and sign, respectively. An atom holds(*DOD-modification-evaluation-info-element1*, *sign1*) specifies that the current evaluations of design object description modifications include the information *DOD-modification-evaluation-info-element1*, with sign *sign1* as its truth value. If *sign1* equals pos, it means that *DOD-modification-evaluation-info-element1* is known to hold (i.e., is true); if *sign1* equals neg, it means that *DOD-modification-evaluation-info-element1* is known to not hold (i.e., is false);); if *sign1* equals unk, it means that it is unknown whether or not *DOD-modification-evaluation-info-element1* holds.

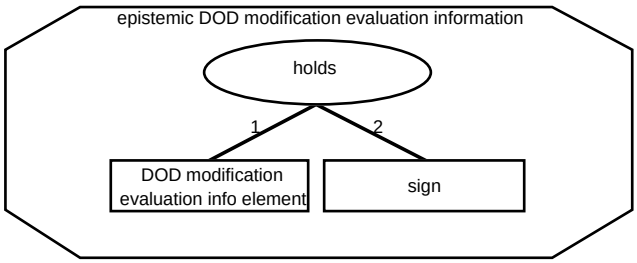


FIGURE 9.30. Structure of information type epistemic DOD modification evaluation information.

Example 9.9.
"It is known that the last design object description alteration, which involved the deletion of an unsatisfactory part, is acceptable."

holds(is-acceptable-DOD-modification-to(
deletion-of(domain-object-information(is-part-of(*frame1*, *bicycle1*), pos)), DOD1), pos)

Figure 9.31 shows the abstraction relations between a DOD modification analysis process and its sub-processes.

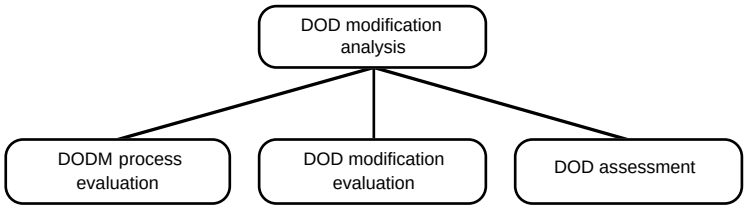


FIGURE 9.31. Two levels of abstraction for a DOD modification analysis process.

Figure 9.32 shows the possibilities for information exchange between processes involved in DOD modification analysis, modelled as information links within the component DOD modification analysis.

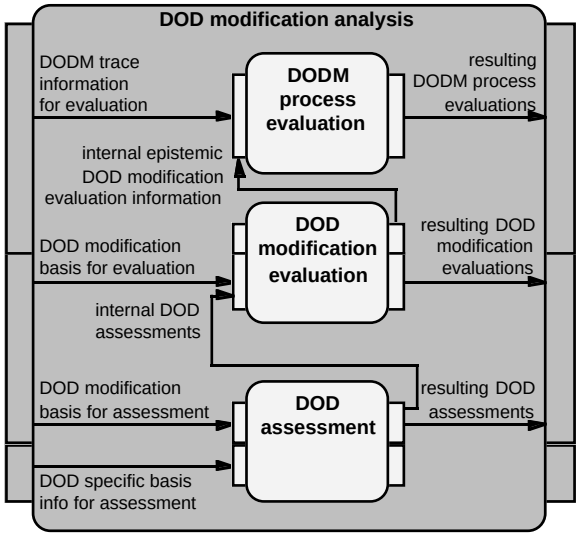


FIGURE 9.32. Information exchange between processes involved in DOD modification analysis.

Table 9.6 presents a possible model of the task control for a DOD modification analysis process, given its sub-processes and possibilities for information exchange. (Note that sequential task control is assumed, but parallel control is equally justifiable.)

TABLE 9.6. Task control for a DOD modification analysis process.

State	*Information link(s) updated next*	*Component(s) activated next*
DOD modification analysis has started.	DODM trace information for evaluation, DOD modification basis for evaluation, DOD modification basis for assessment, and DOD specific basis info for assessment.	DOD assessment.
DOD assessment has terminated.	Internal DOD assessments.	DOD modification evaluation.
DOD modification evaluation has terminated.	Internal epistemic DOD modification evaluation information.	DODM process evaluation.
DODM process evaluation has terminated.	Resulting DODM process evaluations, resulting DOD modification evaluations, and resulting DOD assessments.	None (DOD modification analysis terminates).

9.2.1.1 A specialisation for DOD assessment

Analogously to RQS assessment, the following processes are involved at the two highest levels of process abstraction of a design object description assessment process:

- *DOD assessment* as a whole, which assesses the current design object description on the basis of the current requirement qualification set,
- *basic DOD assessment preparation*, which sets targets and assumptions for the assessment of the current design object description (to determine whether it satisfies the design requirements included in the current requirement qualification set),
- *basic DOD assessment*, which assesses the current design object description to determine whether it satisfies specific design requirements,
- *basic DOD assessment completion*, which establishes a relation between the current requirement qualification set and given epistemic information about basic assessments of the current design object description, and
- *overall DOD assessment*, which assesses the current design object description to determine whether it fulfils the current requirement qualification set.

Figure 9.33 shows the types of information that the sub-processes of a DOD assessment process use as input and the types of information that these processes produce as output. Note that most of these information types have already been introduced in Chapter 8; the remaining information types are explained below.

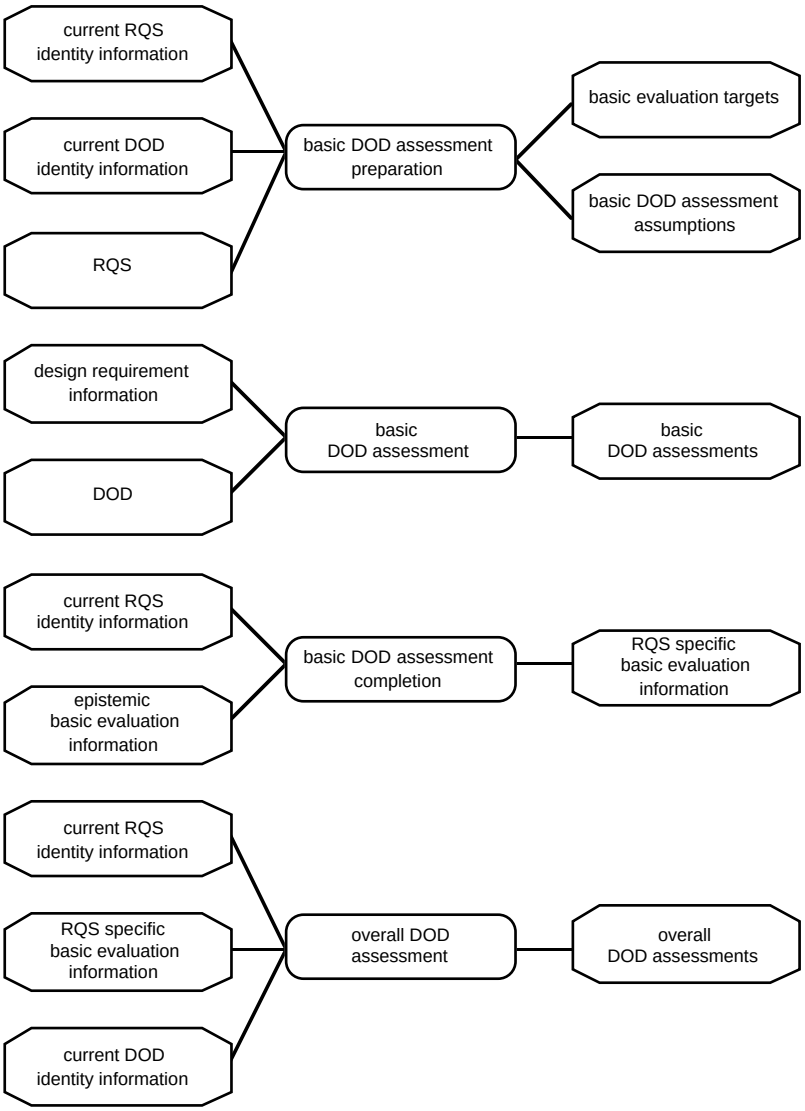


FIGURE 9.33. Input and output of sub-processes of DOD assessment.

Figure 9.34 shows the structure of the information type basic DOD assessment assumptions, which models assumptions about the current design requirements to be used for basic assessment of the current design object description. It contains the relation is-to-be-assumed, which has two arguments of the sorts design-requirement-info-element and sign, respectively. An atom is-to-be-assumed(*design-requirement-info-element1*, *sign1*) specifies that the information *design-requirement-info-element1*, with sign *sign1* as its assumed value, is a current assumption for basic DOD assessment. If *sign1* equals pos, it means that *design-requirement-info-element1* is assumed to be true; if *sign1* equals neg, it means that *design-requirement-info-element1* is assumed to be false.

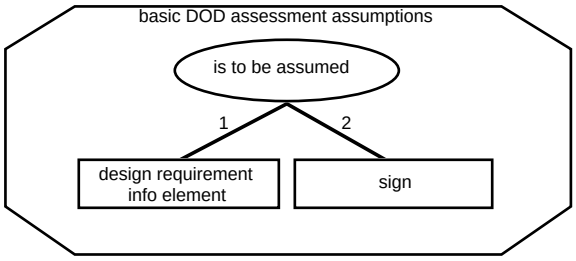


FIGURE 9.34. Structure of information type basic DOD assessment assumptions.

Example 9.10.
"One assumption is that the requirement named R1 states that the bicycle to be designed must be suited for riding in mountains. Another assumption is that the qualified requirement named QR1 states that it is necessary to satisfy R1."

is-to-be-assumed(is-defined-as(*R1*, *is-suitable-for-terrain-type*(*bicycle1*, *mountains*)), pos)
is-to-be-assumed(is-defined-as(*QR1*, every, [*R1*]), pos)

Figure 9.35 shows the abstraction relations between a DOD assessment process and its sub-processes.

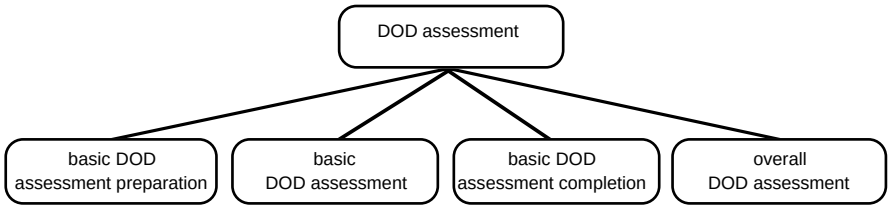


FIGURE 9.35. Two levels of abstraction for a DOD assessment process.

Figure 9.36 shows the possibilities for information exchange between processes involved in DOD assessment, modelled as information links within the component DOD assessment. Note that DOD assessment is positioned at the first and second meta-level of a design process.

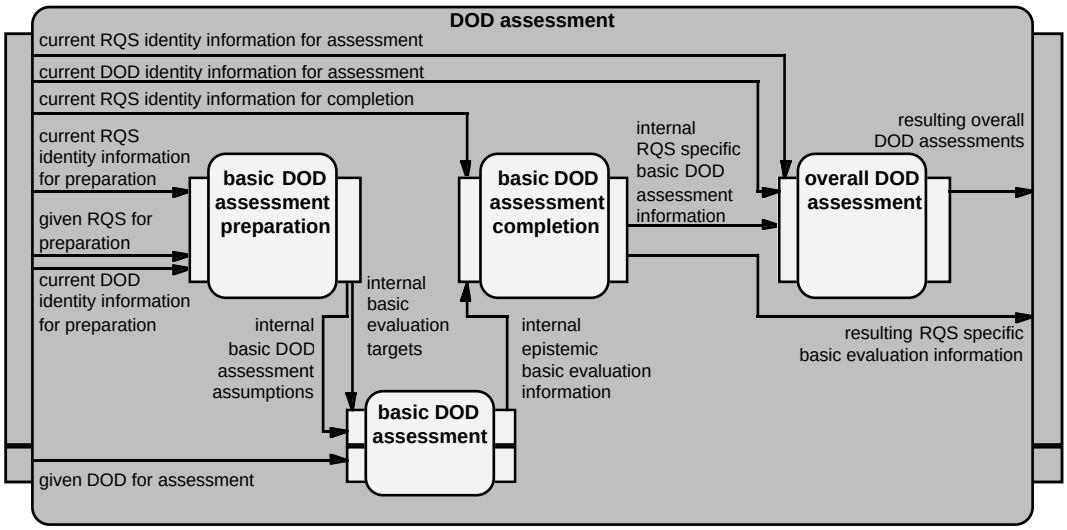


FIGURE 9.36. Information exchange between processes involved in DOD assessment.

Table 9.7 presents a possible model of the task control for a DOD assessment process, given its sub-processes and possibilities for information exchange.

TABLE 9.7. Task control for a DOD assessment process.

State	*Information link(s) updated next*	*Component(s) activated next*
DOD assessment has started.	Current RQS identity information for preparation, current DOD identity information for preparation, and given RQS for preparation.	Basic DOD assessment preparation.
Basic DOD assessment preparation has terminated.	Internal basic evaluation targets, internal basic DOD assessment assumptions, and given DOD for assessment.	Basic DOD assessment.
Basic DOD assessment has terminated.	Current RQS identity information for completion, and internal epistemic basic evaluation information.	Basic DOD assessment completion.
Basic DOD assessment completion has terminated.	Current RQS identity information for assessment, current DOD identity information for assessment, and internal RQS specific basic evaluation information.	Overall DOD assessment.
Overall DOD assessment has terminated.	Resulting overall DOD assessments, and resulting RQS specific basic evaluation information.	None (DOD assessment terminates).

The remainder of this section describes the knowledge used by the sub-processes of a DOD assessment process. (For the sake of brevity, the description is not specified in detail.)

Knowledge for basic DOD assessment preparation. Preparation for basic DOD assessment involves the knowledge that it has to be determined for each design requirement included in the current requirement qualification set whether or not it is satisfied by the current design object description. Preparation also involves the knowledge that all design requirements included in the current requirement qualification set have to be assumed to hold for design requirement assessment. This knowledge is modelled by the following rules within the knowledge base of the component basic DOD assessment preparation:

```
if is-current-DOD(CurDN: DOD-name)
  and is current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(R: requirement-name, E: domain-object-info-expression), pos)
then is-to-be-determined(
    is-decisive-wrt-satisfaction-of(CurDN: DOD-name, R: requirement-name), pos);

if is-current-DOD(CurDN: DOD-name)
  and is current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(R: requirement-name, E: domain-object-info-expression), pos)
then is-to-be-assumed(
    is-defined-as(R: requirement-name, E: domain-object-info-expression), pos);

if is-current-DOD(CurDN: DOD-name)
  and is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
then is-to-be-determined(
    is-decisive-wrt-satisfaction-of(CurDN: DOD-name, QR: qualified-requirement-name), pos);

if is-current-DOD(CurDN: DOD-name)
  and is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
then is-to-be-assumed(
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos);
```

Knowledge for basic DOD assessment. Basic DOD assessment involves knowledge to determine whether or not a given design object description satisfies a specific design require-

ment. This knowledge (which is used within the component basic DOD assessment) is explained informally hereafter.

Given that the definition of a requirement involves a domain object information expression, a design object description satisfies a requirement if and only if it satisfies the domain object information expression defined by the requirement (given a design object domain theory). A design object description violates a requirement if and only if it satisfies the negation of the domain object information expression defined by the requirement (given a design object domain theory). Finally, a design object description is indecisive with respect to the satisfaction of a requirement if and only if it satisfies neither the domain object information expression nor the negation of that same expression defined by the requirement (given a design object domain theory).

Given that the definition of a qualified requirement involves a qualification and a list of requirements, a design object description satisfies a qualified requirement if and only if it satisfies the right combination of requirements from the requirements list defined by the qualified requirement, given the qualification of that list. A design object description violates a qualified requirement if and only if it cannot satisfy the right combination of requirements from the requirements list defined by the qualified requirement, given the qualification of that list. Finally, a design object description is indecisive with respect to the satisfaction of a qualified requirement if and only if it is indecisive with respect to the satisfaction of some requirements from the requirements list defined by the qualified requirement, as a result of which it cannot be decided to agree with the qualification of that list.

Knowledge for basic DOD assessment completion. Completion of basic DOD assessment involves the knowledge that each element of basic evaluation information that currently holds, a relation has to be established to the current requirement qualification set. This knowledge (which is the same as for design requirement assessment completion) is modelled by the following rule within the knowledge base of the component basic DOD assessment completion:

```
if is-current-RQS(CurRN: RQS-name)
  and holds(E: basic-evaluation-info-element, S: sign)
then includes-basic-evaluation-information(
  CurRN: RQS-name, E: basic-evaluation-info-element, S: sign);
```

Knowledge for overall DOD assessment. Overall DOD assessment involves knowledge to determine whether or not a given design object description fulfils a specific requirement qualification set. This knowledge (used within the component overall DOD assessment) is explained informally hereafter.

A design object description fulfils a requirement qualification set if and only if it satisfies each of the set's design requirements that are to be satisfied. A design object description fails to fulfil a requirement qualification set if and only if it violates one or more of the set's de-

sign requirements that are to be satisfied. Finally, a design object description is indecisive with respect to the fulfilment of a requirement qualification set if and only if it satisfies some of the set's design requirements that are to be satisfied and is indecisive with respect to the satisfaction of the set's other design requirements that are to be satisfied.

9.2.1.2 DOD modification evaluation

The task of a DOD modification evaluation process is to evaluate DOD alterations. This evaluation may involve the results of assessing the current design object description, in order to determine the acceptability of a specific design object description alteration. For example, it may be that the last alteration is accepted only if it has succeeded to remove a violation of one of the design requirements that are to be satisfied.

9.2.1.3 DODM process evaluation

The task of an DODM process evaluation process is to evaluate the current design object description manipulation process on the basis of a specific overall design strategy. This evaluation may be performed using generic, domain-specific or heuristic knowledge. Using generic knowledge requires overall design strategies to be formulated in terms of non-subjective criteria. Using domain-specific knowledge or heuristic knowledge (e.g., stating the maximum number of attempts to be tried) requires a thorough understanding of how design object description manipulation processes take place in specific application domains.

9.2.2 A specialisation for DOD modification determination

As for RQS modification determination, the specialisation for DOD modification determination is based on the "object-act" paradigm. According to this specialisation, the following processes are involved at the two highest levels of process abstraction of DOD modification determination:

- *DOD modification determination* as a whole,
- *DOD modification focus determination*, which determines the part of the current design object description for which a modification is to be determined next,
- *DOD modification method determination*, which determines the method by means of which a modification to the current focus is to be determined next,
- *DOD modification method execution*, which executes the current method in order to determine a modification to the current focus.

Figure 9.37 shows the types of information that the sub-processes of a DOD modification determination process use as input and produce as output. Note that these information types have already been introduced in Chapter 8, with the exception of DOD modification focus and current modification method identity information.

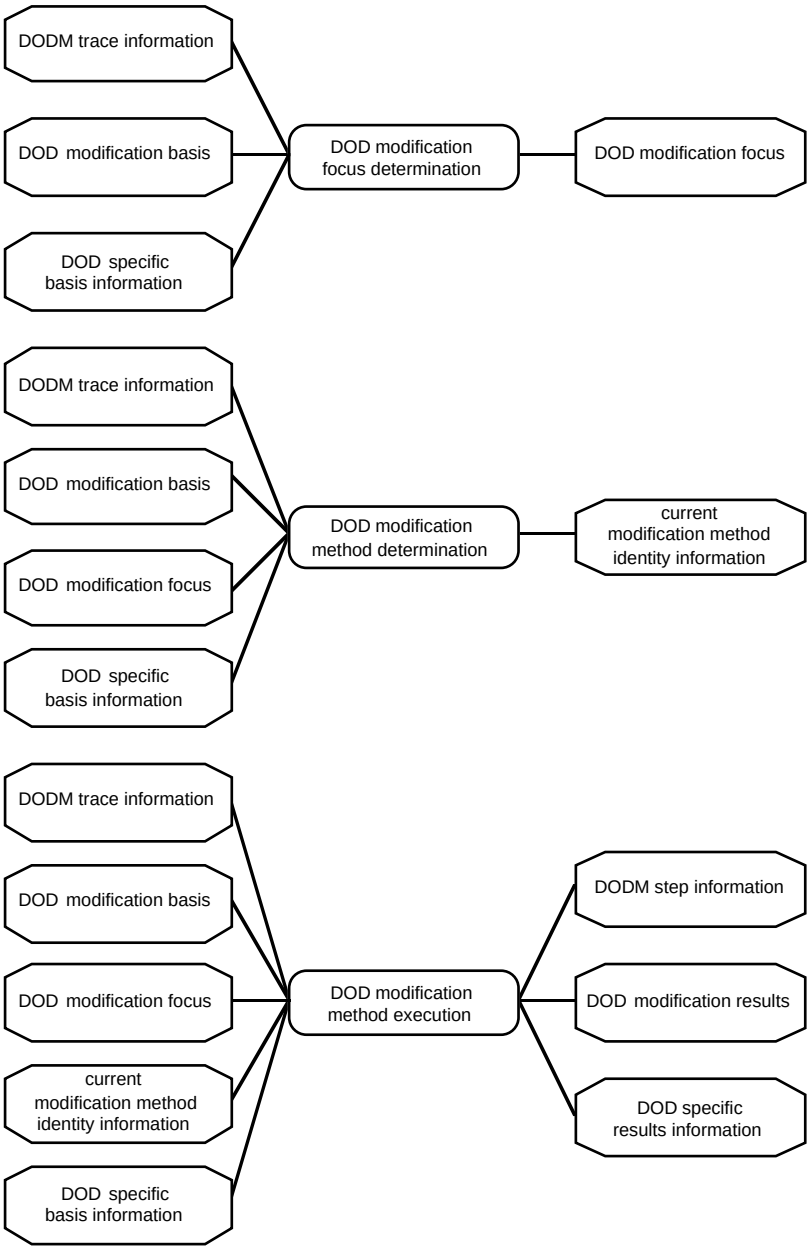


FIGURE 9.37. Input and output of sub-processes of DOD modification determination.

Figure 9.38 shows the structure of the information type DOD modification focus, which models information about the part of a design object description (i.e., a sub-set of the domain object information) that is is currently in focus. It contains the relation is-in-current-DOD-modification-focus, that has one argument of the sort domain-object-info-literal. An atom is-in-current-DOD-modification-focus(*domain-object-info-literal1*) specifies that the domain object in-

formation *domain-object-info-literal1* included in the current design object description is a current subject of modification.

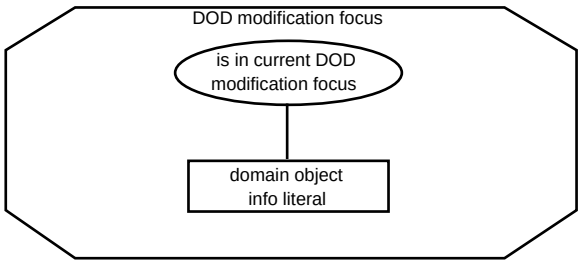


FIGURE 9.38. Structure of information type DOD modification focus.

Example 9.11.
"Current subjects of design object description modification are the information about the bicycle's frame and the information about the motion transfer system."

is-in-current-DOD-modification-focus(
  domain-object-information(is-part-of(*frame1*, *bicycle1*), pos))
is-in-current-DOD-modification-focus(
  domain-object-information(is-part-of(*motion-transfer-system1*, *bicycle1*), pos))
is-in-current-DOD-modification-focus(
  domain-object-information(*has-number-of-gears*(*motion-transfer-system1*, 6), pos))

The structure of the information type current modification method identity information has been explained earlier to model information about which RQS modification method currently applies. For the purpose of DOD modification, this information type can be used as follows.

Example 9.12.
"The current method for modification is to extend the information about the bicycle's frame with new domain object information, in order to produce a more complete picture of that specific part."

is-current-modification-method(*extension*)

Figure 9.39 shows the abstraction relations between a DOD modification determination process and its sub-processes.

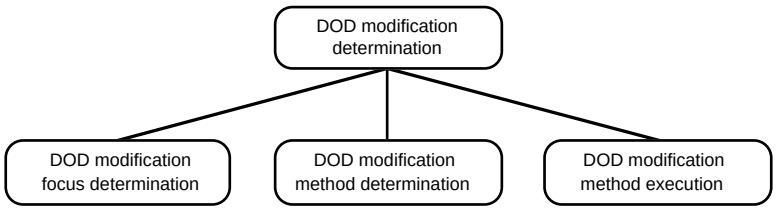


FIGURE 9.39. Two levels of abstraction for a DOD modification determination process.

Figure 9.40 shows the possibilities for information exchange between processes involved in DOD modification determination, modelled as information links within the component DOD modification determination. Note that DOD modification determination is positioned at the first, second and third meta-level of a design process.

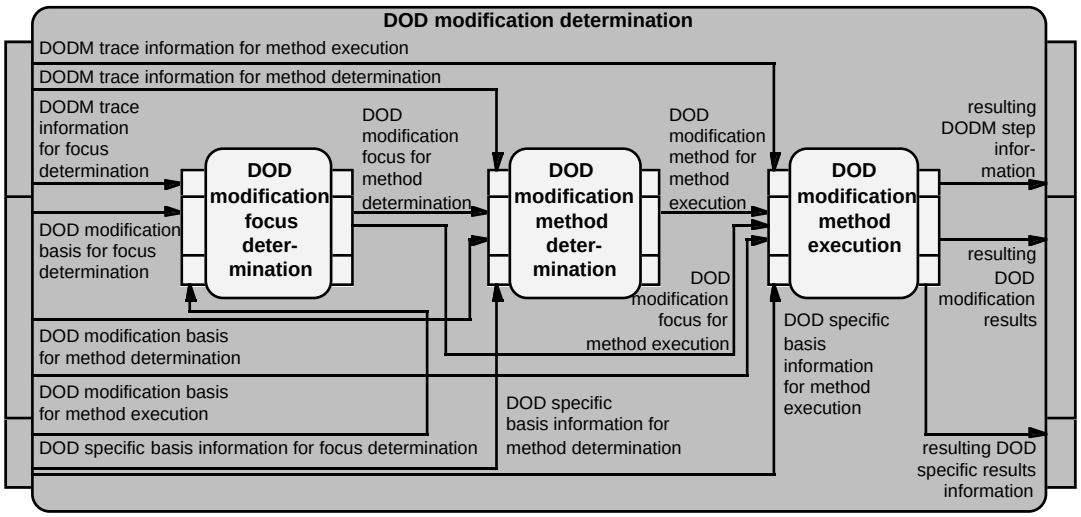


FIGURE 9.40. Information exchange between processes involved in DOD modification determination.

Table 9.8 presents a possible model of the task control for a DOD modification determination process, given its sub-processes and possibilities for information exchange.

TABLE 9.8. Task control for a DOD modification determination process.

State	*Information link(s) updated next*	*Component(s) activated next*
DOD modification determination has started.	DODM trace information for focus determination, DOD modification basis for focus determination, and DOD specific basis information for focus determination.	DOD modification focus determination.

State	*Information link(s) updated next*	*Component(s) activated next*
DOD modification focus determination has terminated.	DOD modification focus for method determination, DODM trace information for method determination, DOD modification basis for method determination, and DOD specific basis information for method determination.	DOD modification method determination.
DOD modification method determination has terminated.	DOD modification focus for method execution, DOD modification method for method execution, DODM trace information for method execution, DOD modification basis for method execution, and DOD specific basis information for method execution.	DOD modification method execution.
DOD modification method execution has terminated.	Resulting DODM step information, resulting DOD modification results, and resulting DOD specific results information.	None (DOD modification determination terminates).

The knowledge used by the three sub-processes of a DOD modification determination process can be expected to be of a domain-specific or heuristic nature. Typically, these three sub-processes will use knowledge about a specific method for the transformation, translation, hierarchical decomposition, reduction or extension of design object descriptions. (This non-exhaustive list of possibilities to determine a modification of a design object description is analogous to the list of possibilities to modify a requirement qualification set as mentioned by Treur [Treur, 1991]).

Chapter 10

Application of GDM to Elevator Configuration

The generic design model GDM has been used to create a model of elevator configuration in the VT domain (Vertical Transportation), where the object to be designed is an elevator and the design requirements consist of customer specifications, building dimensions and constraints. The VT model has been used to automatically generate a prototype software system for elevator configuration in the VT domain.

Publications. *This chapter is based on an earlier publication about the development of a model of VT elevator configuration [Brazier, Langen, Treur, Wijngaards and Willems, 1996].*

One advantage of using a generic design model is that it supports the analysis and modelling of a specific type of design process in a specific application domain. To develop GDM and study its applicability, several types of design processes have been analysed and modelled, among which the process of elevator configuration in the VT domain (Vertical Transportation) [Marcus, Stout and McDermott, 1988; Marcus and McDermott, 1989; Yost and Rothenfluh, 1996]. This chapter describes a specialisation of GDM for this type of design process.

This chapter is organised as follows. Section 10.1 presents the process composition of a VT elevator configuration process, Section 10.2 the knowledge composition and Section 10.3 the relation between the process composition and the knowledge composition. These sections are based on GDM and the specialisations presented in Chapter 9, with the focus on specialising GDM for the VT domain. Section 10.4 presents excerpts of a sample trace, produced by a prototype software system automatically generated from the specification of VT in DESIRE. Finally, Section 10.5 discusses the usability of GDM in modelling VT.

10.1 Process Composition

This section describes processes involved in a VT elevator configuration process at different levels of abstraction as well as the composition of these processes.

10.1.1 Processes at different abstraction levels

This sub-section describes processes involved in a VT elevator configuration process and the different levels of abstraction at which these processes play a role.

10.1.1.1 Processes

For the VT domain, Yost and Rothenfluh describe a system that must be able to make a complete elevator configuration on the basis of given customer specifications, building dimensions and constraints [Yost and Rothenfluh, 1996]. A configuration consists of parts (such as the hoistway and the car assembly) and parameters (such as the hoistway pit depth and the door model). The system must be able to print a description of a configuration.

A VT elevator configuration process is clearly a design process. The design object is an elevator, which is subject to requirements that are formed by customer specifications, building dimensions and constraints. The goal of the process is to generate a complete elevator configuration with no constraint violations.

In the following, VT elevator configuration processes are described in terms of the processes modelled by GDM, together with the types of input information they use and the types of output information they produce.

Design as a whole

A VT elevator configuration process generates a complete elevator configuration on the basis of customer specifications and building dimensions, such that no constraints are violated. For such a design process, no design process objectives are given.

The input and output information types of design as distinguished in GDM were used as follows to model the input and output of a VT elevator configuration process. The input information type design process objectives and the output information type design process evaluations were both not used. The input/output information type RQS was used to model customer specifications and constraints, and the input/output information type DOD was used to model elevator configurations. The output information type RQS assessments was used to model assessments of customer specifications and constraints, and the output information type DOD assessments was used to model assessments of elevator configurations in relation to customer specifications and constraints.

Design process co-ordination
The overall strategy pursued by a VT elevator configuration process is a fixed plan of activities [Yost and Rothenfluh, 1996]:

1. accept customer specifications and building dimensions,
2. derive a first elevator configuration with preliminary assignments of parts and values for parameters,
3. check for constraint violations,
4. propose and implement configuration modifications until a complete elevator configuration with no constraint violations is devised, and
5. print a description of the final configuration.

The input and output information types of design process co-ordination as distinguished in GDM were used as follows to model the input and output of overall strategy determination in the VT domain. The input information type design process objectives and the output information type design process evaluations were both not used. The input information type control process evaluations was used to model information about the advances in accepting and modifying customer specifications, building dimensions and constraints as well as information about the advances in deriving, checking and modifying elevator configurations. The output information type overall design strategy was used to model the aforementioned strategy.

Requirement qualification set manipulation
VT's requirement qualification set manipulation process accepts customer specifications and building dimensions, derives constraints and, if necessary, revises customer specifications to make it possible to devise a satisfactory elevator configuration. In the context of VT, the subprocesses of a requirement qualification set manipulation process as distinguished in Chapter 7 were interpreted as follows:

- the *RQS modification* process accepts customer specifications and building dimensions and, if a satisfactory elevator configuration is deemed infeasible, it modifies customer specifications;
- the *RQS modification history maintenance* process records the history and rationale of deriving constraints, accepting customer specifications and building dimensions and modifying customer specifications during an elevator configuration process;
- the *deductive RQS refinement* process deduces standard constraints of an elevator configuration;
- the *current RQS maintenance* process keeps track of the contents of the current set of customer specifications, building dimensions and constraints.

The sub-processes of an RQS modification process as distinguished in Chapter 9 were interpreted as follows:

- the *RQS modification analysis* process analyses the results of modifying the current set of customer specifications, building dimensions and constraints. It determines whether the last modification has resulted in a complete set of customer specifications, building dimensions and constraints, as well as whether the last modification introduced untenable requirements;
- the *RQS modification determination* process generates (at the start of the design process) a proposal to accept customer specifications and building dimensions or (for untenable requirements) a proposal to modify specific customer specifications.

These two processes are composed of more fine-grained processes; for the details, see [Brazier, Langen, Treur, Wijngaards and Willems, 1996].

Design object description manipulation

VT's design object description manipulation process determines a complete elevator configuration on the basis of given customer specifications, building dimensions and constraints. In the context of VT, the sub-processes of a design object description manipulation process as distinguished in Chapter 8 were interpreted as follows:

- the *DOD modification* process generates an initial elevator configuration with preliminary assignments of parts and values for parameters, it checks for constraint violations, and it proposes and implements elevator configuration modifications until a complete elevator configuration with no constraint violations is devised;
- the *DOD modification history maintenance* process records the history and rationale of proposing, implementing and undoing modifications to elevator configurations;
- the *deductive DOD refinement* process deduces those parts and values for parameters of an elevator configuration that depend on parts already known and on parameters of which the values are already known;
- the *current DOD maintenance* process keeps track of the contents of the current elevator configuration.

The sub-processes of a DOD modification process as distinguished in Chapter 9 were interpreted as follows:

- the *DOD modification analysis* process analyses the results of modifying the current elevator configuration. It determines whether the last modification has resulted in a complete configuration that does not violate any constraints and, if the previous configuration violated a constraint, whether the last modification fixed that constraint violation without introducing new violations;
- the *DOD modification determination* process proposes initial or revised parts and values of parameters of an elevator configuration. Revision takes place according to the following procedure: first select the next violated constraint to be resolved, then de-

termine a combination of (iterative or non-iterative) fixes that could resolve the selected constraint violation, and finally determine the current iteration step of each of the iterative fixes involved in the selected combination.

These two processes are composed of more fine-grained processes; for the details, see [Brazier, Langen, Treur, Wijngaards and Willems, 1996].

10.1.1.2 Process abstraction levels

Being a design process, the upper levels of process abstraction of a VT elevator configuration process are the same as distinguished in GDM and its specialisations described in Chapter 9. For details of the lower levels, see [Brazier, Langen, Treur, Wijngaards and Willems, 1996].

10.1.2 Composition of processes

This section describes the way in which the higher-level processes involved in a VT elevator configuration process are composed of lower-level processes, in terms of possibilities for exchange of information between processes and in terms of task control knowledge used to control both the processes and the information exchange.

10.1.2.1 Information exchange

Being a design process, the exchange of information between the upper-level processes involved in a VT elevator configuration process is the same as distinguished in GDM and its specialisations described in Chapter 9. For details about the exchange of information between the lower-level processes, see [Brazier, Langen, Treur, Wijngaards and Willems, 1996].

10.1.2.2 Task control

Being a design process, the knowledge to control the upper-level processes involved in a VT elevator configuration process is the same as distinguished in GDM and its specialisations described in Chapter 9. For details about the knowledge to control the lower-level processes, see [Brazier, Langen, Treur, Wijngaards and Willems, 1996].

10.2 Knowledge Composition

This section describes the knowledge structures involved in a VT elevator configuration process at different levels of reflection. First the levels of reflection are described and then the structure and composition of the knowledge at each of these reflection levels.

10.2.1 Reflection levels

In the context of VT, the four reflection levels distinguished in GDM are interpreted as follows:

- the *object level* includes information about the values of parts and parameters of an elevator configuration. This level also includes knowledge about the VT domain.

- the *first meta-level* includes information about customer specifications and building dimensions and about elevator configurations. This level also includes deductive refinement knowledge to deduce standard constraints.

- the *second meta-level* includes information about sets of customer specifications, building dimensions and constraints. This level also includes strategic knowledge to determine modifications of customer specifications and to determine modifications of elevator configurations, respectively.

- the *third meta-level* includes strategic knowledge to determine a fixed overall strategy for elevator configuration.

10.2.2 Knowledge structures and composition

This section describes the composition and the structure of the knowledge related to a VT elevator configuration process and shows how they are modelled in DESIRE. For the domain-specific knowledge structures, the VT ontology by Gruber, Olsen and Runkel has been used, which is expressed in KIF/Ontolingua [Gruber, Olsen and Runkel, 1996], in addition to the description of the VT knowledge by Yost and Rothenfluh [Yost and Rothenfluh, 1996].

Note that the composition and contents of an ontology is influenced by the type of process for which it is used. The type of process determines not only which information about the domain is to be modelled but also the terminology to be employed. In the context of VT, for example, colour is irrelevant and therefore not included in the VT ontology, whereas characteristics and dimensions of elevator components are described in terms of parameters, which is motivated by the view of an elevator as a configuration. These observations agree with the argument that domain knowledge cannot be adequately represented independent of the type of processes for which it has been designed [Vanwelkenhuysen and Mizoguchi, 1995].

10.2.2.1 Structure and composition of knowledge at the object level

The object level includes information and knowledge about the parts and parameters of an elevator configuration.

Information types

The main organisational structure used is a concept hierarchy, which conceptualises the components of an elevator (e.g., the car assembly) and part-of relations between these components (e.g., between a passenger cab, a supporting structure, and a safety on one hand and a car assembly on the other hand). Figure 10.1 depicts part of the VT concept hierarchy.

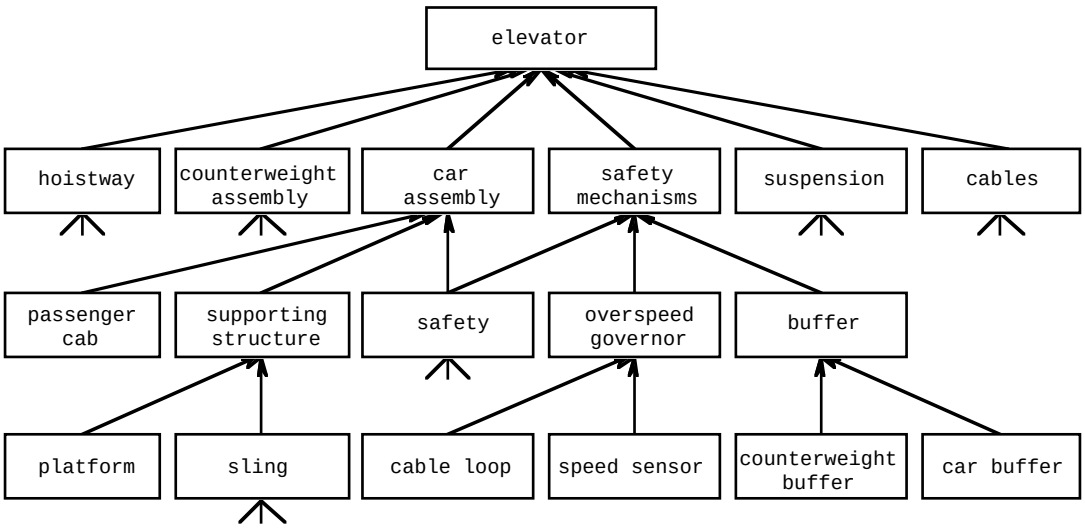


FIGURE 10.1. Part of the VT concept hierarchy (arrows denote part-of relations).

The second type of structure used is a concept-attribute-value structure. A car assembly, for example, has a number of attributes such as weight, guideshoe weight and supplement weight with real numbers as values. In the VT ontology, a concept and an attribute together uniquely identify a parameter of the elevator configuration; for example, the concept *door* and its attribute *model* together identify the parameter *door model.* In an elevator configuration, each parameter may have one single value (or none, if the configuration is not yet complete). Figure 10.2 shows part of the concept-attribute-value structure for a car assembly.

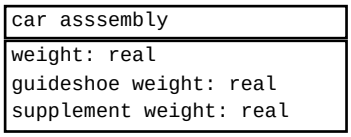


FIGURE 10.2. Part of the concept-attribute-value structure for a car assembly.

In the VT ontology, the value of a parameter is defined to be of a specific type. Figure 10.3 shows the four types of value that can be assigned to parameters:

- **integer** for values of parameters such as *opening count* (defined as the number of floors the elevator will stop on);
- **real** for values of parameters such as *hoistway pit depth* (defined as the depth in inches of the hoistway pit);
- **string** for values of parameters such as *door model* (defined for side-opening doors to consist of two codes separated by a dash);
- **boolean** for values of parameters such as *car lantern* (defined as an indication of whether or not the car should be equipped with a lantern).

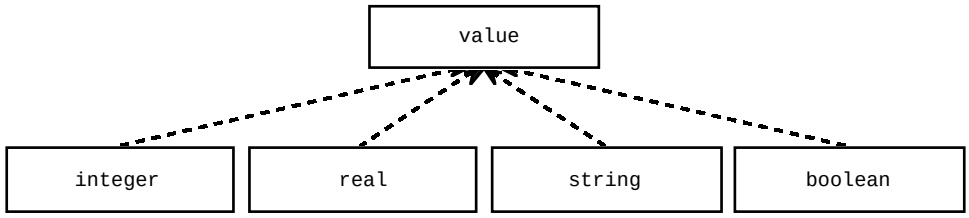


FIGURE 10.3. Value type hierarchy (arrows denote is-a relations).

The set of parameters and their values describe a (partial) elevator configuration. Of specific interest are the values that the parameters have in the current configuration that is under construction or being examined.

To refine GDM for the VT domain, GDM's information type application-specific-domain-object-information was extended with references to three application specific information types: parameter-definitions, parameter-expression-evaluations and current-parameter-value-information.

```
information type application-specific-domain-object-information
  information types
    parameter-definitions,
    parameter-expression-evaluations,
    current-parameter-value-information;
end information type
```

Parameters were modelled by objects of the sort parameter. (Note that a choice has been made to model instances of parameters as objects; an alternative would have been to include sorts for concepts and attributes and to define functions from combinations of concepts and

attributes to parameters.) For the sake of brevity, the specification below does not enumerate all parameters distinguished for the VT domain, but only four of them.

information type parameter-type
sorts
parameter

objects
car-weight,
door-speed,
hoistway-depth,
sling-model, …: parameter;
end information type

The general value type and the four more specific value types from the value type hierarchy were modelled as sorts and the is-a relations between them by sub-sort relations. (For the sake of brevity, it is not explained here how to model integers, reals, strings and booleans.)

information type value-type
information types
integer-type, real-type,
string-type, boolean-type;

sorts
value

subsorts
integer,
real,
string,
boolean: value;
end information type

Expressions of parameters (involving parameters, constant values and functions such as sum, difference, product and quotient) were modelled within the information type parameter-definitions by a sort parameter-expression with sub-sorts parameter and value.

information type parameter-expression-type
information types
parameter-type,
value-type;

```
  sorts
    parameter-expression

  subsorts
    parameter,
    value: parameter-expression;

  functions
    sum, difference, product, quotient,
    …: parameter-expression * parameter-expression -> parameter-expression;
end information type
```

Within the information type parameter-definitions, a relation is used to model definitions of parameters, and another relation is used to model information about the order of string values for a given parameter.

```
information type parameter-definitions
  information types
    parameter-expression-type,
    value-type;

  relations
    is-defined-as: parameter * parameter-expression;
    is-defined-to-have-as-lowest-value: parameter * value;
    is-defined-to-have-as-value-order: parameter * value * value;
end information type
```

Information about the actual values that parameters have within an elevator configuration was modelled by a relation within the information type current-parameter-value-information.

```
information type current-parameter-value-information
  information types
    parameter-type, value-type;

  relations
    has-as-current-value: parameter * value;
end information type
```

Information about the values corresponding to parameter expressions within an elevator configuration was modelled by a relation within the information type parameter-expression-evaluations.

information type parameter-expression-evaluations
information types
parameter-expression-type, value-type;

relations
evaluates-to-value: parameter-expression * value;
end information type

Knowledge bases
The knowledge base involved in the deductive DOD refinement process is composed of two other knowledge bases: a knowledge base with definitions of parameters, and a knowledge base with rules of deduction for the values of parameters on the basis of their definitions.

Knowledge to define parameters. The values of some of the parameters of an elevator configuration are uniquely defined. Among the parameters, dependent parameters and constant parameters are distinguished. *Dependent parameters* have values that are fully determined by those of one or more other parameters. For example, the *sling underbeam* and the *counterweight stack height* (abbreviated cwt stack height) are defined by:

$$\textit{sling underbeam} = \textit{car cab height} + \textit{sling underbeam space} \quad (10.1)$$
$$\textit{cwt stack height} = \textit{cwt plate quantity} \cdot \textit{individual cwt plate thickness} \quad (10.2)$$

where '+' and '·' denote sum and product, respectively.

Constant parameters are parameters of which the value is a constant, rather than determined by other parameters. For example, the *car buffer footing channel height* is defined by:

$$\textit{car buffer footing channel height} = 3.5 \quad (10.3)$$

where the value signifies a height measurement in inches.

In more abstract terms, the VT ontology assumes that the object-level domain knowledge can be modelled in terms of equations of the form $y = f(\underline{x})$, together with pre-conditions stating in which situations these equations apply. In such an equation, y denotes a parameter of which the value is defined to be equal to a (numerical) function f of the values of a tuple $\underline{x}$ of other parameters. By application of such an equation, the value of y can be determined from the values of the parameters $\underline{x}$; if y is a dependent parameter, then $\underline{x}$ is non-empty, whereas if y is a constant parameter, then $\underline{x}$ is empty.

In the VT ontology, the possible values of string parameters are defined by means of tables. At the start of an elevator configuration process, most of these parameters are assigned the lowest possible value, which may be changed later if necessary. For example, the possible values for the *crosshead model*, ordered from low to high, are: W8x18, W8x21, C8x11.5, C10x15.3 and C13x16.55.

The definitions of parameters were modelled as facts within the application specific knowledge base parameter-definition-knowledge of the generic component deductive-DOD-refinement:

```
is-defined-as(sling-underbeam, sum(car-cab-height, sling-underbeam-space));
is-defined-as(cwt-stack-height, product(cwt-plate-quantity, cwt-plate-thickness));
is-defined-as(car-buffer-footing-channel-height, 3.5);
is-defined-to-have-as-lowest-value(crosshead-model, W8x18);
is-defined-to-have-as-value-order(crosshead-model, W8x18, W8x21);
is-defined-to-have-as-value-order(crosshead-model, W8x21, C8x11.5);
is-defined-to-have-as-value-order(crosshead-model, C8x11.5, C10x15.3);
is-defined-to-have-as-value-order(crosshead-model, C10x15.3, C13x16.55);
```

Knowledge to deduce actual parameter values. Knowledge about the format of parameter value definitions is used to deduce the actual value of a dependent parameter. The values of parameters in the definition of a dependent parameter may be determined by definitions (which means a recursive step) or retrieved from the current elevator configuration.

Assuming a configuration process with definitions of parameters in the form of formulas (involving sum, difference, product and quotient, for instance), the knowledge to deduce the actual value of a dependent parameter or a constant parameter was modelled by the following fact and rules within the application specific knowledge base parameter-value-deduction-knowledge of the generic component deductive-DOD-refinement:

```
evaluates-to-value(V: value, V: value);

if has-as-current-value(P: parameter, V: value)
then evaluates-to-value(P: parameter, V: value);

if evaluates-to-value(E1: parameter-expression, V1: value)
  and evaluates-to-value(E2: parameter-expression, V2: value)
  and V3: value = V1: value + V2: value
then evaluates-to-value(
    sum(E1: parameter-expression, E2: parameter-expression), V3: value);

if evaluates-to-value(E1: parameter-expression, V1: value)
  and evaluates-to-value(E2: parameter-expression, V2: value)
  and V3: value = V1: value – V2: value
then evaluates-to-value(
    difference(E1: parameter-expression, E2: parameter-expression), V3: value);
```

```
if evaluates-to-value(E1: parameter-expression, V1: value)
  and evaluates-to-value(E2: parameter-expression, V2: value)
  and V3: value = V1: value * V2: value
then evaluates-to-value(
    product(E1: parameter-expression, E2: parameter-expression), V3: value);

if evaluates-to-value(E1: parameter-expression, V1: value)
  and evaluates-to-value(E2: parameter-expression, V2: value)
  and not V2: value = 0
  and V3: value = V1: value / V2: value
then evaluates-to-value(
    quotient(E1: parameter-expression, E2: parameter-expression), V3: value);

if is-defined-as(P: parameter, E: parameter-expression)
  and evaluates-to-value(E: parameter-expression, V: value)
then has-as-current-value(P: parameter, V: value);
```

10.2.2.2 Structure and composition of knowledge at the first meta-level

The first meta-level includes information and knowledge about elevator configurations, customer specifications and constraints, and parameter value defaults and modifications.

Information types

The VT ontology contains concepts for elevator configurations, customer specifications and constraints, and parameter value modifications.

Elevator configurations. In a VT elevator configuration process, a distinction is made between a *current configuration* (i.e., the most recent elevator configuration that resulted from accepted modifications) and a *tentative configuration* (i.e., the most recent elevator configuration that resulted from modifications to the current configuration that have yet to be accepted or rejected). By defining the sort elevator-configuration as a sub-sort of the generic sort DOD-name, GDM's information type DOD-name-type was instantiated to the VT domain.

```
information type DOD-name-type
  information types
    elevator-configuration-type;

  sorts
    DOD-name
```

subsorts
elevator-configuration: DOD-name;
end information type

The current configuration and the tentative configuration were modelled as objects of the sort elevator-configuration.

information type elevator-configuration-type
sorts
elevator-configuration

objects
current-configuration, tentative-configuration: elevator-configuration;
end information type

Customer specifications and constraints. In a VT elevator configuration process, design requirements appear in the form of customer specifications and constraints. For example, the customer has to specify the car capacity (i.e., the maximum total car occupant weight, in pounds, that the elevator must be able to support, which is a number between 2000 and 4000), and one of the constraints is that the maximum hoistway bracket spacing is 168 inches. The remainder of this section shows how constraints in particular are modelled.

A *constraint* imposes a limit on the possible values of a specific parameter (which is referred to as the parameter on which the constraint focuses). Four different types of constraint are distinguished, as shown in Figure 10.4:

- A **min constraint** imposes a lower limit on the value of a numeric parameter. For example, constraint C-6 is a min constraint on the car overtravel:

  $$\textit{car overtravel} \geq \textit{cwt runby} + 1.5 \cdot \textit{cwt buffer stroke} + 24 \qquad (10.4)$$

- A **max constraint** imposes an upper limit on the value of a numeric parameter. For example, constraint C-22 is a max constraint on the counterweight stack height:

  $$\textit{cwt stack height} \leq \textit{cwt frame height} - \textit{cwt frame thickness} \qquad (10.5)$$

- A **range constraint** imposes a lower and upper limit on the value of a numeric parameter. For example, constraint C-2 is a range constraint on the car cab height:

  $$84 \leq \textit{car cab height} \leq 240 \qquad (10.6)$$

- A **compatibility constraint** enumerates the possible values of a string parameter that are compatible with the values of other parameters. For example, constraint C-34 is a compatibility constraint on the motor model:

$$\textit{machine model} = 18 \Rightarrow \textit{motor model} = \text{10HP} \vee \textit{motor model} = \text{15HP} \qquad (10.7)$$

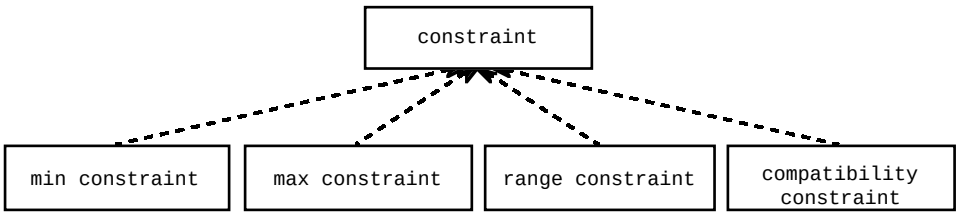


FIGURE 10.4. Constraint type hierarchy (arrows denote is-a relations).

By defining the sort constraint as a sub-sort of the generic sort requirement-name, GDM's information type requirement-name-type was instantiated to the VT domain.

information type requirement-name-type
information types
constraint-type;

sorts
requirement-name

subsorts
constraint: requirement-name;
end information type

The general constraint type and the four more specific constraint types from the constraint type hierarchy were modelled by sorts and the is-a relations between them by sub-sort relations.

information type min-constraint-type
sorts
min-constraint

objects
min-car-overtravel, min-platform-to-hoistway-left,

```
    min-hoist-cable-safety-factor, …: min-constraint;
end information type

information type max-constraint-type
  sorts
    max-constraint

  objects
    max-hoistway-bracket-spacing, max-machine-groove-pressure,
    max-car-supplement-weight, …: max-constraint;
end information type

information type range-constraint-type
  sorts
    range-constraint

  objects
    range-car-cab-height, range-car-buffer-load, range-car-buffer-quantity, …: range-constraint;
end information type

information type compatibility-constraint-type
  sorts
    compatibility-constraint

  objects
    compatible-hoist-cable-quantity, compatible-motor-model,
    compatible-platform-model: compatibility-constraint;
end information type

information type constraint-type
  information types
    min-constraint-type, max-constraint-type, range-constraint-type, compatibility-constraint-type;

  sorts
    constraint

  subsorts
    min-constraint, max-constraint, range-constraint, compatibility-constraint: constraint;
end information type
```

By including a reference to the information type constraint-focus-information, GDM's information type application-specific-design-requirement-information was instantiated to the VT domain.

information type application-specific-design-requirement-information
information types
constraint-focus-information;
end information type

Information about the parameter on which a specific constraint focuses was modelled by a relation within the information type constraint-focus-information.

information type constraint-focus-information
information types
constraint-type, parameter-type;

relations
focuses-on: constraint * parameter;
end information type

Parameter value modifications. In a VT elevator configuration process, there a few ways to modify the value of a parameter. The value of a numeric parameter (e.g., the *opening height*) may be increased or decreased, the extent of which is specified by an expression consisting of constants and parameters (e.g., *opening height – car cab height*). The value of a string parameter (e.g., the *machine groove model*) may be upgraded, which means that a value directly higher than the current value is to be selected (according to the defined order of values).

By including a reference to the information type parameter-value-modifications, GDM's information type application-specific-DOD-information was instantiated to the VT domain.

information type application-specific-DOD-information
information types
parameter-value-modifications;
end information type

Information about the current parameter value modifications to be applied was modelled by relations within the information type parameter-value-modifications.

information type parameter-value-modifications
information types
parameter-expression-type;

```
  relations
    is-to-be-upgraded: parameter;
    is-to-be-increased-by, is-to-be-decreased-by: parameter * parameter-expression;
end information type
```

Knowledge bases

The knowledge bases involved on the first meta-level of an elevator configuration process are a knowledge base for the deduction of constraints, a knowledge base for the deduction of default parameter values, and a knowledge base for the determination of design object description modifications to be applied.

Constraint deduction. There are fifty standard constraints for the VT domain. For example, C-11 (alias max-compensation-cable-load-car-side-car-top) is a max constraint on the compensation cable load on the car side of the machine sheave when the car is at the top:

$$\textit{platform model} = 2.5\text{B} \Rightarrow \textit{compensation cable load car side car top} \leq 600 \qquad (10.8)$$

This constraint was modelled by the following fact within the domain-specific knowledge base of the generic component deductive-RQS-refinement:

```
is-defined-as(max-compensation-cable-load-car-side-car-top,
  for-all(Var1, l-implies(l-and(has-as-current-value(platform-model, 2.5B),
      has-as-current-value(compensation-cable-load-car-side-car-top, Var1)), Var1 ≤ 600)));
```

Deduction of default parameter values. A VT elevator configuration process uses default values for some of the parameters. (At the start of an elevator configuration process, VT's DOD modification determination process uses knowledge about defaults to determine initial values of these parameters; these values may be changed later in the VT elevator configuration process, if necessary.) For example, the following defaults are defined:

$$\textit{sling model} = 2.5\text{B-}18 \qquad (10.9)$$

$$\textit{cwt between guiderails} = 28 \qquad (10.10)$$

$$\textit{cwt runby} = 0.01 \cdot \textit{hoistway travel} + 6 \qquad (10.11)$$

Using GDM's generic relations includes-domain-object-information and has-default-domain-object-information, this knowledge was modelled by the following facts and rule within the application specific knowledge base of (a sub-component of) the component DOD-modification-determination:

```
has-default-domain-object-information(
  N: DOD-name, has-as-current-value(sling-model, 2.5B-18), pos);
```

```
has-default-domain-object-information(
  N: DOD-name, has-as-current-value(cwt-between-guiderails, 28), pos);

if includes-domain-object-information(
    N: DOD-name, has-as-current-value(hoistway-travel, V1: value), pos)
  and V2: value = 0.01 * V1: value + 6
then has-default-domain-object-information(
    N: DOD-name, has-as-current-value(cwt-runby, V2: value), pos);
```

Determination of design object description modifications. The current elevator configuration is to be modified according to the selected parameter value modification. If a parameter is to be increased (or decreased) by a given value, then the new parameter value is determined by evaluating the parameter expression defined by the parameter value modification and adding the resulting value to (or subtracting the result from) the current parameter value. If a parameter is to be upgraded, then the new parameter value is determined by selecting the value directly higher than the parameter's current value (according to the defined order of values for the parameter). Regardless of the way in which the parameter value is to be modified, the current parameter value is to be retracted.

Using GDM's generic relations includes-domain-object-information, is-to-be-asserted and is-to-be-retracted, this knowledge was modelled by the following facts and rules within an application specific knowledge base of the component DOD-modification-determination:

```
if is-current-DOD(CurDN: DOD-name)
  and is-to-be-increased-by(P: parameter, E: parameter-expression)
  and includes-domain-object-information(
    CurDN: DOD-name, has-as-current-value(P: parameter, V: value), pos)
then is-to-be-retracted(has-as-current-value(P: parameter, V: value), pos);

if is-current-DOD(CurDN: DOD-name)
  and is-to-be-decreased-by(P: parameter, E: parameter-expression)
  and includes-domain-object-information(
    CurDN: DOD-name, has-as-current-value(P: parameter, V: value), pos)
then is-to-be-retracted(has-as-current-value(P: parameter, V: value), pos);

if is-current-DOD(CurDN: DOD-name)
  and is-to-be-upgraded(P: parameter)
  and includes-domain-object-information(
    CurDN: DOD-name, has-as-current-value(P: parameter, V: value), pos)
then is-to-be-retracted(has-as-current-value(P: parameter, V: value), pos);
```

```
if is-current-DOD(CurDN: DOD-name)
  and is-to-be-increased-by(P: parameter, E: parameter-expression)
  and includes-domain-object-information(
    CurDN: DOD-name, has-as-current-value(P: parameter, V1: value), pos)
  and includes-domain-object-information(
    CurDN: DOD-name, evaluates-to-value(E: parameter-expression, V2: value), pos)
  and V3: value = V1: value + V2: value
then is-to-be-asserted(has-as-current-value(P: parameter, V3: value), pos);

if is-current-DOD(CurDN: DOD-name)
  and is-to-be-decreased-by(P: parameter, E: parameter-expression)
  and includes-domain-object-information(
    CurDN: DOD-name, has-as-current-value(P: parameter, V1: value), pos)
  and includes-domain-object-information(
    CurDN: DOD-name, evaluates-to-value(E: parameter-expression, V2: value), pos)
  and V3: value = V1: value – V2: value
then is-to-be-asserted(has-as-current-value(P: parameter, V3: value), pos);

if is-current-DOD(CurDN: DOD-name)
  and is-to-be-upgraded(P: parameter)
  and includes-domain-object-information(
    CurDN: DOD-name, has-as-current-value(P: parameter, V1: value), pos)
  and includes-domain-object-information(
    CurDN: DOD-name, is-defined-to-have-as-value-order(P: parameter, V1: value, V2: value),
pos)
then is-to-be-asserted(has-as-current-value(P: parameter, V2: value), pos);
```

10.2.2.3 Structure and composition of knowledge at the second meta-level

The second meta-level includes information and knowledge about sets of elevator design requirements, about parameter value initialisations, assessments and fixes.

Information types

The VT ontology contains concepts for fixes to constraint-violating parameters. These concepts are related to the modification of elevator configurations.

Fixes. In a VT elevator configuration process, fixes are distinguished for resolving detected violations of constraints. There may be multiple fixes for one constraint. For a specific constraint, a fix proposes a modification of the value of the corresponding parameter. If it is a numeric parameter, a modification means to increase or decrease the value by a given value, specified in the fix. If it is a string parameter (which models a motor model, for instance), a

modification means to upgrade the value (i.e., to change the value to the next higher predefined value).

It may be that a fix only applies in specific situations; the applicability conditions of a fix specify what the contents of the current elevator configuration should be for the fix to apply. Some fixes are more desirable than others: fixes that have little impact on the contents of the elevator configuration are preferred to those that have a greater impact. The ten desirability codes that are distinguished range from D1 ("No problem") to D10 ("Changes major contract specifications"), where the last is least preferred.

A fix is a potential part of a modification if it applies to the constraint that is currently being processed (i.e., the violated constraint currently in focus) and if the applicability condition of the fix holds for the current elevator configuration. Whether a potential fix becomes part of the next modification to the current elevator configuration depends on the fixes tried before for the same violated constraint, and on the preference level of the fix. Single fixes are preferred to combinations of fixes at the same preference level and fixes at lower preference levels are preferred to fixes at higher preference levels.

Some of the fixes are applied iteratively; for example, one of the fixes to constraint C-3 (defining the allowed car buffer blocking heights) is "to increase the hoistway pit depth by one-inch steps, as long as constraint C-3 is violated." How many steps are to be applied next in an iterative fix depends on the number of steps that have already been tried in the same fix and on the other, currently tried fixes for the same constraint, if any.

A fix may not be acceptable, as it may fail to remove a violation of a constraint or it may unwantedly introduce new violations of other constraints. If a fix is not acceptable, the tentative configuration (resulting from modifying the current configuration) will be discarded.

By including references to the information types fix-definitions, fix-constraint-relations, fix-preferences, fix-applicability-conditions, constraint-identity-information, fix-eligibility-information, fix-identity-information and fix-acceptability-information, GDM's information type application-specific-DOD-alteration-information was instantiated to the VT domain.

information type application-specific-DOD-alteration-information
  information types
    fix-definitions, fix-constraint-relations, fix-applicability-conditions, fix-preferences,
    constraint-identity-information, fix-eligibility-information,
    fix-identity-information, fix-acceptability-information;
end information type

Fixes and parameter value modifications were modelled by sorts, and definitions of fixes by a relation within the information type fix-definitions, using GDM's generic sort DOD-specific-results-info-element.

```
information type fix-type
  sorts
    fix

  objects
    fix3-1, fix3-2, fix4-1, fix4-2, fix5, … : fix;
end information type

information type fix-definitions
  information types
    fix-type, DOD-specific-results-info-element-type;

  relations
    is-defined-as: fix * DOD-specific-results-info-element;
end information type
```

Preferences of fixes were modelled by a sort, and information about the preference levels of different fixes was modelled by a relation within the information type fix-preferences.

```
information type preference-type
  information types
    preference-type;

  sorts
    preference

  objects
    D1, D2, D3, D4, D5, D6, D7, D8, D9, D10: preference;
end information type

information type fix-preferences
  information types
    fix-type, preference-type;

  relations
    has-preference: fix * preference;
end information type
```

Information about the constraint on which a fix focuses was modelled by a relation within the information type fix-constraint-relations.

information type fix-constraint-relations
information types
fix-type, constraint-type;

relations
focuses-on-constraint: fix * constraint;
end information type

Using the sort fix and GDM's generic sort domain-object-info-expression, information about the condition under which a fix applies was modelled by a relation within the information type fix-applicability-conditions.

information type fix-applicability-conditions
information types
fix-type, domain-object-info-expression-type;

relations
is-applicable-under-condition: fix * domain-object-info-expression;
end information type

Information about the constraint currently processed and the constraint previously processed was modelled by a relation within the information type constraint-identity-information.

information type constraint-identity-information
information types
constraint-type;

relations
is-previously-processed, is-currently-processed: constraint;
end information type

Information about potential fixes and most preferred fixes was modelled by relations within the information type fix-eligibility-information.

information type fix-eligibility-information
information types
fix-type;

relations
is-potential-fix, is-most-preferred-fix: fix;
end information type

Information about previous and current fixes and fix steps (using integers to model the number of steps) was modelled by relations within the information type fix-identity-information.

```
information type fix-identity-information
  information types
    fix-type, integer-type;

  relations
    is-previous-fix,
    is-current-fix: fix;
    is-previous-fix-step,
    is-current-fix-step: fix * integer;
end information type
```

Information about the acceptability of fixes was modelled by relations within the information type fix-acceptability-information.

```
information type fix-acceptability-information
  information types
    fix-type;

  relations
    is-acceptable-fix: fix;
end information type
```

Knowledge bases

This section presents parts of the knowledge bases for the processes of analysing and determining modifications of elevator configurations.

DOD modification analysis. VT's DOD modification analysis process determines which modifications to the elevator configuration are acceptable and which are not. Application of a fix to the current configuration yields a tentative configuration. The fix is not acceptable if, in the tentative configuration, (a) the constraint that was processed is still violated, or (b) a constraint on one of the modified values is violated. This knowledge was modelled by the following rules within the knowledge base of the component DOD-modification-evaluation:

```
if is-current-RQS(CurRN: RQS-name)
  and is-current-DOD(CurDN: DOD-name)
  and is-previously-processed(C: constraint)
  and includes-basic-evaluation-information(
    CurRN: RQS-name, violates(CurDN: DOD-name, C: constraint), pos)
```

```
  and is-resulting-DOD(ResDN: DOD-name)
  and includes-basic-evaluation-information(
    CurRN: RQS-name, violates(ResDN: DOD-name, C: constraint), pos)
  and is-previous-fix(F: fix)
then not is-acceptable-fix(F: fix);

if is-current-RQS(CurRN: RQS-name)
  and is-current-DOD(CurDN: DOD-name)
  and holds(includes-domain-object-information(
      CurDN: DOD-name, has-as-current-value(P: parameter, V1: value), pos), pos)
  and is-resulting-DOD(ResDN: DOD-name)
  and holds(includes-domain-object-information(
      ResDN: DOD-name, has-as-current-value(P: parameter, V2: value), pos), pos)
  and not V1: value = V2: value
  and includes-design-requirement-information(
    CurRN: RQS-name, focuses-on(C: constraint, P: parameter), pos)
  and includes-basic-evaluation-information(
    CurRN: RQS-name, violates(ResDN: DOD-name, C: constraint), pos)
  and is-previous-fix(F: fix)
then not is-acceptable-fix(F: fix);
```

DOD modification determination. The knowledge bases involved in VT's DOD modification determination process are a knowledge base for the initialisation of parameter values, a knowledge base for the deduction of properties of fixes, and a knowledge base for the determination of a fix (or fix combination) to be applied next.

VT's DOD modification determination process starts by proposing initial values of parameters. This knowledge was modelled by the following rule within the method-specific knowledge base of the component DOD-modification-method-execution:

```
if is-current-DOD(CurDN: DOD-name)
  and is-current-modification-method(initialise-parameter-values)
  and holds(has-default-domain-object-information(
    CurDN: DOD-name, E: domain-object-info-element, S: sign), pos)
then is-selected-DOD-alteration(
    addition-of(domain-object-information(E: domain-object-info-element, S: sign)));
```

To undo constraint violations, VT's DOD modification determination process applies fixes that revise parameter values. For example, one of the fixes for constraint C-22 (alias max-counterweight-stack-height) is to increase the counterweight plate depth by half-inch steps; this fix is always applicable (i.e., under any condition) and has desirability code D3.

This knowledge was modelled by the following facts within the domain-specific knowledge base of the component DOD-modification-method-execution:

```
is-defined-as(fix-22-1, is-to-be-increased-by(counterweight-plate-depth, 0.5));
focuses-on-constraint(fix-22-1, max-counterweight-stack-height);
is-applicable-under-condition(fix-22-1, l-true);
has-preference(fix-22-1, D3);
```

To determine which of the fixes to apply next is a complex procedure. For the sake of brevity, the text hereafter focuses on the identification of potential fixes.

In the first phase of fix determination, the potential fixes are identified. A fix is a potential fix in a given situation if it focuses on the violated constraint that is currently processed and if the fix is applicable in the given situation (which is determined by the contents of the current elevator configuration). This knowledge was modelled by the following rule within the method-specific knowledge base of the component DOD-modification-method-execution:

```
if is-current-RQS(CurRN: RQS-name)
  and is-current-DOD(CurDN: DOD-name)
  and is-currently-processed(C: constraint)
  and includes-basic-evaluation-information(
    CurRN: RQS-name, violates(CurDN: DOD-name, C: constraint), pos)
  and is-current-modification-method(fix-constraint-violations)
  and focuses-on-constraint(F: fix, C: constraint)
  and is-applicable-under-condition(F: fix, E: domain-object-info-expression)
  and holds(covers(CurDN: DOD-name, E: domain-object-info-expression), pos)
then is-potential-fix(F: fix);
```

10.2.2.4 Structure and composition of knowledge at the third meta-level

The third and highest meta-level includes information and knowledge about overall elevator design strategies.

Information types

There are many possible strategies for a VT elevator configuration process, one of which is to follow a propose-and-revise approach. This overall strategy means that the requirement qualification set manipulation process has to propose an initial design requirements specification (consisting of customer specifications and constraints) and to revise the requirements specification if the customer, installation contractors and building inspector do not approve the proposed elevator configuration. Furthermore, this overall strategy means that the design object description manipulation process has to propose initial values for the parameters of the elevator configuration and then to revise values until no constraints are violated. A revision is

acceptable if and only if it resolves the violation of the constraint being processed and does not introduce new constraint violations.

By defining the sort VT-strategy-name as a sub-sort of the generic sort design-strategy-name, GDM's information type design-strategy-name-type was instantiated to the VT domain.

information type design-strategy-name-type
information types
VT-strategy-name-type;

sorts
design-strategy-name

subsorts
VT-strategy-name: design-strategy-name;
end information type

The strategy to follow a propose-and-revise approach was modelled by an object of the sort VT-strategy-name.

information type VT-strategy-name-type
sorts
VT-strategy-name

objects
propose-and-revise: VT-strategy-name;
end information type

Knowledge bases

For a VT elevator configuration process, Yost and Rothenfluh describe one overall strategy, propose-and-revise, which applies to every stage of the elevator configuration process and involves that an initial solution is to be proposed and revised until it is satisfactory [Yost and Rothenfluh, 1996]. This knowledge was modelled by the following fact within the knowledge base of the component DPC:

includes-overall-design-strategy(S: design-process-state, propose-and-revise);

10.3 Relation between Compositions of Process and Knowledge

Being a design process, the relation between the process composition and the knowledge composition of a VT elevator configuration process is the same as for GDM and its specialisations described in Chapter 9.

10.4 Sample Trace

This section presents excerpts from a sample trace produced by a prototype system that has been automatically generated from the full DESIRE specification of the VT model. Parts of this specification have been presented in Section 10.2. The test case described by Yost and Rothenfluh [Yost and Rothenfluh, 1996] has been fully reproduced with this prototype system.

10.4.1 Trace of violated constraints and fixes

For the test case provided by Yost and Rothenfluh [Yost and Rothenfluh, 1996], Table 10.1 shows the violated constraints that were detected during the design process, together with the fixes that were applied to remove these violations. The table shows the violated constraints, the proposed fix (or fix combination), and whether or not the fix (combination) was accepted.

By means of an example, it is shown how to read Table 10.1. The fifth row states that the value of the parameter *cwt-to-platform-rear* has been decreased in seven steps by the amount of 0.5 (so the decrease was 0.5 in the first step and 3.5 in the seventh step), in an unsuccessful attempt to resolve the violation of the constraint *max-traction-ratio*. The eighth row states that, in order to resolve the violation of the same constraint, the value of the parameter *car-supplement-weight* has been increased in five steps by the amount of 100. With each such step, the value of the parameter *cwt-to-platform-rear* has been decreased in seven steps by the amount of 0.5.

TABLE 10.1. Violated constraints and fixes.

Violated constraint(s)	*Fix (or fix combination)*			*Accept*
	Focus parameter	*Step*	*Action*	
Min-platform-to-hoistway-left	*Opening-to-hoistway-left*	1	Increase by (8 – *platform-to-hoistway-left*)	Yes
Eligible-motor-model	*Machine-model*	1	Upgrade	Yes
Max-vertical-rail-force	*Car-railunit-weight*	1	Upgrade	Yes
Min-hoist-cable-safety-factor	*Hoist-cable-quantity*	1	Increase by (5 – *hoist-cable-quantity*)	Yes
Max-traction-ratio	*Cwt-to-platform-rear*	1-7	Decrease by 0.5	No

Violated constraint(s)	*Fix (or fix combination)*			*Accept*
	Focus parameter	*Step*	*Action*	
Max-traction-ratio, min-cwt-to-platform-rear	*Car-supplement-weight*	1-6	Increase by 100	No
Max-traction-ratio, max-car-supplement-weight	*Car-supplement-weight*	1	Increase by 100	No
	Cwt-to-platform-rear	1	Decrease by 0.5	
Max-traction-ratio	*Car-supplement-weight*	1-5	Increase by 100	No
	Cwt-to-platform-rear	1-7	Decrease by 0.5	
Max-traction-ratio, min-cwt-to-platform-rear	*Car-supplement-weight*	6	Increase by 100	No
	Cwt-to-platform-rear	1	Decrease by 0.5	
Max-traction-ratio, max-car-supplement-weight	*Comp-cable-model*	1	Upgrade	No
Max-traction-ratio	*Comp-cable-model*	1	Upgrade	No
	Cwt-to-platform-rear	1-8	Decrease by 0.5	
Max-traction-ratio, min-cwt-to-platform-rear	*Comp-cable-model*	1	Upgrade	No
	Car-supplement-weight	1-6	Increase by 100	
Max-traction-ratio, max-car-supplement-weight	*Comp-cable-model*	1	Upgrade	No
	Car-supplement-weight	1-4	Increase by 100	
	Cwt-to-platform-rear	1-7	Decrease by 0.5	
Max-traction-ratio	*Comp-cable-model*	1	Upgrade	No
	Car-supplement-weight	5	Increase by 100	
	Cwt-to-platform-rear	1-6	Decrease by 0.5	
Max-traction-ratio	*Comp-cable-model*	1	Upgrade	Yes
	Car-supplement-weight	5	Increase by 100	
	Cwt-to-platform-rear	7	Decrease by 0.5	
Min-machine-beam-section-modulus	*Machine-beam-model*	1	Upgrade	Yes

10.4.2 Trace of component activations

In this section, rather than showing the activations of all the components of the VT model, two examples are presented. The examples illustrate different dynamic aspects of VT.

The first example, presented in Table 10.2, illustrates the co-operation between DPC, DODM and RQSM. The constraint min-platform-to-hoistway-left is violated and a fix to resolve this violation is proposed. However, this fix is not immediately accepted, as it changes a value protected by a requirement. Instead, RQSM is activated and the customer is asked to accept or reject the proposed change. The customer agrees and the requirement is changed accordingly, after which DODM is allowed to continue.

TABLE 10.2. Trace of component activations and results for fixing the min-platform-to-hoistway-left constraint.

Component	*Results*
DOD-modification-analysis	The constraint min-platform-to-hoistway-left is violated.
DOD-modification-determination	To fix the constraint min-platform-to-hoistway-left, parameter *opening-to-hoistway-left* is to be increased by (8 – *platform-to-hoistway-left*).
Current-DOD-maintenance	A (tentative) configuration, including the design object information that the parameter *opening-to-hoistway-left* has a value of 33.
DOD-modification-analysis	The constraint min-platform-to-hoistway-left is now satisfied, and no new constraint violations have been introduced. However, the requirement on the parameter *opening-to-hoistway-left* is not satisfied, so a dead end is reached.
DPC	Because no further progress can be made in devising a configuration, the current set of elevator design requirements has to be modified.
RQS-modification-analysis	The requirement that the parameter *opening-to-hoistway-left* must have a value of 32 is untenable.
RQS-modification-determination	The selected method to determine a new requirement on the parameter *opening-to-hoistway-left* is to consult the customer, with the result that: • the requirement that the parameter *opening-to-hoistway-left* must have a value of 32 is to be deleted, • the requirement that the parameter *opening-to-hoistway-left* must have a value of 33 is to be added.
Current-RQS-maintenance	A new set of elevator design requirements, including design requirement information about a requirement that the parameter *opening-to-hoistway-left* must have a value of 33.
DPC	Because the current set of design requirements has been modified, further attempts must be made to devise an elevator configuration.
DOD-modification-analysis	The last design modification is acceptable.
Current-DOD-maintenance	The values of all dependent parameters which have not yet been recomputed have been removed from the current configuration.
Deductive-DOD-refinement	The values of all dependent parameters have been deduced (again).

The second example, presented in Table 10.3, focuses on resolving the violation of the constraint max-traction-ratio, which activity illustrates various activations of components within DODM. To resolve this constraint violation, several fix combinations have to be tried. For the test case described by Yost and Rothenfluh [Yost and Rothenfluh, 1996], resolving this constraint violation is the most extensive part of the elevator configuration process, as a combination of three fixes is needed. (The table only shows a small fragment, as the complete activation sequence of sub-components of DOD modification within DODM is quite extensive.)

TABLE 10.3. Trace of component activations and results for fixing the max-traction-ratio constraint.

Component	*Results*
DOD-modification-analysis	The constraint max-traction-ratio is violated.
DOD-modification-determination	To fix the constraint max-traction-ratio, parameter *cwt-to-platform-rear* is to be decreased by 0.5.
Current-DOD-maintenance	A (tentative) configuration, including the design object information that the parameter *cwt-to-platform-rear* has a value of 4.75.
DOD-modification-analysis	The constraint max-traction-ratio is violated and the fix last tried is not acceptable, so the tentative configuration is not accepted. However, further fix steps for the given fix are still possible.
DOD-modification-determination	To fix the constraint max-traction-ratio, parameter *cwt-to-platform-rear* is to be decreased by 1.0 (i.e., two times 0.5).
…	…
DOD-modification-determination	To fix the constraint max-traction-ratio, parameter *cwt-to-platform-rear* is to be decreased by 3.5 (i.e., seven times 0.5).
Current-DOD-maintenance	A (tentative) configuration, including the design object information that the parameter *cwt-to-platform-rear* has a value of 1.75.
DOD-modification-analysis	The constraints max-traction-ratio and min-cwt-to-platform-rear are violated and the fix last tried is not acceptable, so the tentative configuration is not accepted. Moreover, further fix steps will never resolve the violation of min-cwt-to-platform-rear.
DOD-modification-determination	No more fix steps are possible on parameter *cwt-to-platform-rear*. To fix the constraint max-traction-ratio, parameter *car-supplement-weight* is to be increased by 100.
And so forth.	…

10.5 Discussion

To model VT, an earlier version of the generic design model GDM was re-used to create a model of elevator configuration and to automatically generate a running prototype (on Unix). No records were kept of the development process, but the effort spent was in the order of a few person-months. Modelling VT has been a useful exercise, resulting in a better understanding of design and improvements to GDM.

To model VT, GDM's process and knowledge composition defined for a design process as a whole could be used as is. Extensions for RQS manipulation and DOD manipulation were needed to model VT's method of proposing and revising customer specifications and elevator configurations. Furthermore, modelling VT led to improvements to GDM with respect to defaults, design history and the exchange of information between RQS manipulation and DOD manipulation.

Also other researchers from Artificial Intelligence have worked on modelling VT and developed running systems that are able to solve VT problems. The results (including ours) are reported in a special issue of the *International Journal of Human-Computer Studies* (Vol. 44, 1996). Below, the most important features of the work of other researchers are discussed.

Yost shows how the VT problem is solved using the Soar/TAQL environment [Yost, 1996]. Soar is a problem-solving architecture that supports a stratified, structure-preserving system-building approach. Soar's underlying computational model is PSCM (*Problem Space Computational Model*), which distinguishes tasks (particular problems to be solved), states (consisting of objects relevant to the task), operators (manipulating and transforming states) and problem spaces (comprising states and operators and the knowledge that relates to them). TAQL (*Task AcQuisition Language*) is a language on top of Soar, which eases the specification of tasks and methods.

The TAQL elevator-configuration system for VT has nine problem spaces, containing 24 operators in all, where six of the nine problem spaces (19 of the operators) are devoted to fixing constraint violations. Building this system took about 35 hours.

Rothenfluh, Gennari, Eriksson, Puerta, Tu and Musen report on the use of the PROTÉGÉ-II framework and tool set to solve the VT problem [Rothenfluh, Gennari, Eriksson, Puerta, Tu and Musen, 1996]. PROTÉGÉ-II is a knowledge-engineering environment that focuses on the use of reusable ontologies and problem-solving methods to generate task-specific knowledge acquisition tools and executable problem solvers.

PROTÉGÉ-II was used to develop a knowledge-based system for VT, ELVIS (*ELeVator configuration In Sisyphus*). In ELVIS, three types of knowledge are distinguished: configuration parameter knowledge (composed of component knowledge and model knowledge), constraint knowledge and fix knowledge. This knowledge is used within the PROTÉGÉ-II implementation of the problem solving method propose-and-revise, which is based on chronological backtracking. Building ELVIS took about 55 hours.

Motta, Stutt, Zdrahal, O'Hara and Shadbolt describe how the VITAL methodology and toolkit was used to solve the VT problem [Motta, Stutt, Zdrahal, O'Hara and Shadbolt, 1996]. The VITAL methodology characterises the development of knowledge-based systems as the construction of four main process products. The requirement specification describes the expected functionalities and eventual constraints of the system, the conceptual model models the expertise required to perform the task, the design models are a series of increasingly more specific system models, and the executable code is the actual system.

To solve the VT problem, an initial model of the problem solving method propose-and-revise was constructed using a generative grammar of model fragments and then refined and operationalised in the VITAL operational conceptual modelling language (OCML). The prototype system was directly compiled from the OCML model and took between 3 and 4 person-months to develop, where 75 per cent of the effort went into the domain analysis.

Schreiber and Terpstra describe how they used CommonKADS to solve the VT problem [Schreiber and Terpstra, 1996]. CommonKADS is based on the idea that various perspectives or models are important in the process of developing a knowledge-based system: models of

the organisation, of the overall task, of the agents involved, of the required communication, of the expertise, and of the design of the final artefact.

The CommonKADS domain-knowledge model of VT distinguishes a task-type oriented ontology for parametric design, a method-oriented ontology for propose-and-revise, and an ontology mapping from the parametric-design ontology onto the propose-and-revise ontology. The CommonKADS task model of VT distinguishes three sub-tasks: propose a design extension, verify the current design, and revise the design, if necessary. The SIADL environment (*SImulated Application Design Language*) was used to design and realise a running prototype for VT. Altogether, modelling VT took a few person-weeks.

Runkel, Birmingham and Balkany show how the DIDS framework and system was used to solve the VT task [Runkel, Birmingham and Balkany, 1996]. DIDS (*Domain-Independent Design System*) enables knowledge engineers to rapidly build knowledge-based systems by providing libraries of reusable problem-solving procedures and knowledge-acquisition procedures. DIDS assists the engineer in constructing a knowledge-level task description that defines knowledge structures and operators, a process model that defines a problem solving method and a knowledge acquisition model that defines suitable knowledge acquisition tools.

The DIDS approach to VT was to start with an existing knowledge-based system for office assignment and modify it to perform the VT task. For VT, most of the problem-solving structures and knowledge acquisition tools used for the office-assignment task were reused. Building the knowledge-based system for VT took about two person-months.

Poeck, Fensel, Landes and Angele used the knowledge specification language KARL and the CRLM (*Configurable Role-Limiting Method*) approach to make a knowledge-based system that is able to solve the VT problem [Poeck, Fensel, Landes and Angele, 1996]. KARL allows a formal and operational specification of knowledge-based systems, and CRLM provides strong knowledge acquisition support and rapid prototyping.

The domain layer of KARL was used to model the VT domain knowledge, and CRLM to configure the propose-and-revise problem solving method for VT (reusing the propose-and-exchange method developed earlier for office assignment). To model the VT domain took a few months, and to configure the problem solving method only a few days.

In conclusion, it can be stated that reuse of ontologies and problem solving methods will most often result in shorter development times compared to building systems from scratch. Developing a model of a specific design process on the basis of GDM in combination with a library with components and knowledge structures may mean a considerable advantage.

Chapter 11

Application of GDM to Environmental Inventory

The generic design model GDM has been used to create a model of environmental inventory, where the object to be designed is a model of a specific industrial process by means of which the environmental impact of this process can be determined, and where the design requirements are conditions on the quality of the environmental impact information about this process. This environmental inventory model has been used to develop a prototype system for environmental inventory of brick and tile fabrication in the Netherlands.

Publications. *This chapter is based on collaborative research with TNO (the Netherlands Organisation for Applied Scientific Research), the results of which are a prototype running on PC and a TNO publication [Langen, Brazier, Diepenmaat and Pulles, 1995].*

To establish national, regional and global environmental policies, governments require reliable information on emission of pollutants. Industrial plants are among the main polluters for many components. Hence, countries collect, aggregate, interpret, and report environmental information about industrial processes.

An *industrial process* is a process of producing (mainly) physical matter to be delivered or sold to customers. It can be characterised in terms of material flow and energy flow, as illustrated in Figure 11.1. An industrial process uses raw material and energy to produce products, and has side-effects in the form of energy use (or releases as cooling water or air), emissions (to air, soil or water) and waste. For example, a brick fabrication process uses river clay, water and additives, and energy in the form of natural gas and electricity; it produces bricks and heat (hot steam) and emits mostly gases, mainly in the form of carbon dioxide.

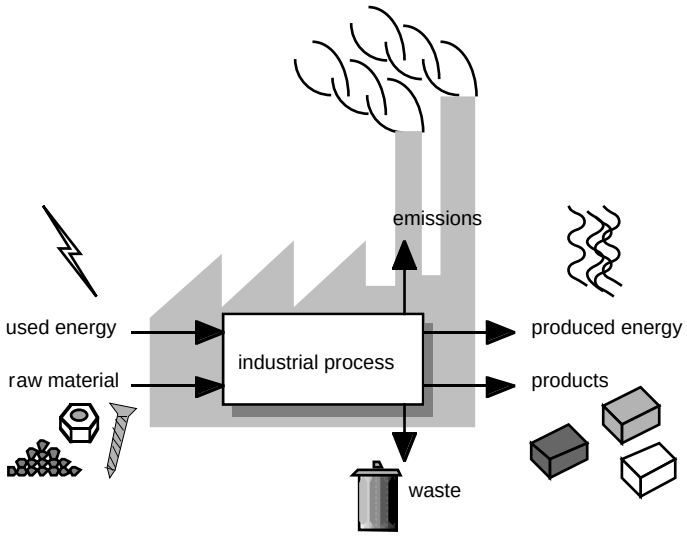


FIGURE 11.1. An industrial process.

The term *environmental impact* is used to denote all effects of an industrial process that a government considers to be harmful to the environment. The production of greenhouse gases, for instance, is considered harmful by most governments. Harmful effects may be caused by raw material usage, energy consumption and energy production, emissions to air and water, and waste. If measurements of such harmful effects are impossible, unreliable, or too expensive, the environmental impact of an industrial process is estimated on the basis of production-related information.

Consider the following example. In the Netherlands, the fabrication of bricks is structured as a chain of four sub-processes: mixing, shaping, drying, and brick-baking (in a brick kiln). For an estimation of the annual carbon dioxide emission of a drying process, the relevant production-related information required is the type and amount of bricks produced per year, the type of fuel used for the dryer and the amount required to dry 1,000 bricks. Suppose, in this example, that the factory of concern produces 30,000,000 red bricks per year, and that drying 1,000 red bricks requires 15 m^3 of natural gas. In combination with environmental chemistry knowledge that in the process of drying red bricks, carbon dioxide is emitted at a rate of 1,770 g per m^3 of natural gas, the annual emission of carbon dioxide by the drying process can be calculated to amount 796,500 kg (almost 800 metric tons).

The Netherlands Organisation for Applied Scientific Research (TNO) is a nationally and internationally acclaimed knowledge and contract-research institute for business and government. Since the 1990s, TNO works on the development of software tools that make it possible for companies and governments to generate environmental impact information about industrial processes. These software tools incorporate generally applicable principles of making environmental inventories and can be extended with software modules for different industrial sectors in different countries.

Company staff and civil servants are not experienced in making environmental inventories, so the software tools provide a means for users to judge whether the quality of the generated environmental impact information is sufficient or not. This approach makes it possible for companies to write their environmental impact reports, and (local) governments to check the validity of those reports, without being experts in environmental chemistry themselves.

For example, within the programme of the European Topic Centre on Air and Climate Change (ETC-ACC) funded by the European Environment Agency, TNO has co-developed a number of software tools for the inventory of air emissions. The CollectER tool supports a user (in practice, a national air emission expert) in collecting national emission inventory data. The ReportER tool supports a user in aggregating, interpreting, and reporting environmental impact information on the basis of national emission inventory data stored in the CollectER database. The EstimatER tools are a family of expert systems, each dedicated to a specific type of industrial process, that contain all necessary information on emission factors and emission estimation methods that are needed to make a quality assured and quality controlled estimate for all relevant emissions to air. (For further information see the web site of ETC-ACC at *http://etc-acc.eionet.eu.int/*. The software tools can be found at the "country support tools" pages of this web site.)

The task of environmental inventory is to make a model of an existing industrial process by means of which the environmental impact of that process can be determined, such that given requirements of the quality of the environmental impact information about that process are satisfied. Hence, environmental inventory is a design process. In the context of TNO's aim to develop suitable software tools for environmental inventory, this chapter describes a specialisation of GDM for environmental inventory.

This chapter is organised as follows. Section 11.1 presents the process composition of an environmental inventory process, Section 11.2 the knowledge composition and Section 11.3 the relation between the process composition and the knowledge composition. These sections are based on GDM and the specialisations presented in Chapter 9, with the focus on specialising GDM for environmental inventory processes. Section 11.4 presents a sample domain, for which a small prototype software system for environmental inventory has been developed: brick and tile fabrication in the Netherlands. Finally, Section 11.5 discusses the usability of GDM in modelling environmental inventory.

11.1 Process Composition

This section describes processes involved in an environmental inventory process at different levels of abstraction as well as the composition of these processes.

11.1.1 Processes at different abstraction levels

This sub-section describes processes involved in an environmental inventory process and the different levels of abstraction at which these processes play a role.

11.1.1.1 Processes

To support environmental inventory processes, TNO opts for an environmental inventory system that generates a model of a specific industrial process, that uses the model to derive environmental impact information about that process, that checks whether the quality of the derived environmental impact information agrees with government standards, and that writes the requested environmental impact report, such as an annual emission report, an environmental balance sheet, a permit application form, or an environmental prospectus ([Langen, Brazier, Diepenmaat and Pulles, 1995]).

An environmental inventory process is a design process. The design object is a model of a specific, existing industrial process from which environmental impact information about the process can be derived, and the design requirements express government standards for the quality of the environmental impact information about that type of process. The goal of the process is to generate a model of an industrial process in such a way that the quality of the derived environmental impact information complies with government standards.

In the following, environmental inventory processes are described in terms of the processes modelled by GDM, together with the types of input information they use and the types of output information they produce.

Design as a whole

An environmental inventory process generates a model of a specific industrial process and derives from the model information about the environmental impact of that process, of which the quality should satisfy the given quality requirements. Usually, a (partial) process model is available at the start of the environmental inventory process (e.g., when an annual emission report for the previous registration year has to be updated for the current year), but sometimes, such a model is not available (e.g., in case a new industrial plant has been built). The industrial process model is developed on the basis of input by a user (e.g., a company staff member) and on defaults for missing production information about the industrial process of concern. The quality of the environmental impact information derived from the process model depends on the quality of the sources used to obtain production information for the industrial process model as well as the methods used to obtain this production information.

The input and output information types of design as distinguished in GDM were used as follows to model the input and output of an environmental inventory process. The input information type design process objectives and the output information type design process evaluations were both not used. The input/output information type RQS was used to model a set of requirements of the quality of the environmental impact information about the type of industrial process that is concerned. The input/output information type DOD was used to model a description of an industrial process, including a (partial) process model and the de-

rived environmental impact information. The output information type RQS assessments was used to model assessments of the quality requirements, and the output information type DOD assessments was used to model assessments of process descriptions in relation to quality requirements.

Design process co-ordination

The overall strategy pursued by an environmental inventory process is a fixed plan of activities [Langen, Brazier, Diepenmaat and Pulles, 1995]:

1. accept quality requirements of the environmental impact information to be generated about the type of industrial process that is concerned,
2. generate a description of the industrial process of concern, which satisfies the given quality requirements,
3. if considered useful, repeat steps 1 and 2 in order to try design requirements imposing a higher quality of the environmental impact information,
4. write the requested environmental impact report.

The input and output information types of design process co-ordination as distinguished in GDM were used as follows to model the input and output of overall strategy determination in an environmental inventory process. The input information type design process objectives and the output information type design process evaluations were both not used. The input information type control process evaluations was used to model information about the advances in establishing quality requirements as well as information about the advances in generating, checking and modifying process descriptions. The output information type overall design strategy was used to model the aforementioned strategy.

Requirement qualification set manipulation

The requirement qualification set manipulation process of an environmental inventory process establishes quality requirements of the environmental impact information about a given industrial process. In the context of environmental inventory, the sub-processes of a requirement qualification set manipulation process as distinguished in Chapter 7 were interpreted as follows:

- the *RQS modification* process determines quality requirements and checks whether these requirements impose a level of quality of environmental impact information that is at least as high as the government standards dictate for the type of industrial process that is concerned;
- the *RQS modification history maintenance* process records the history and rationale of accepting and modifying quality requirements during environmental inventory;

- the *deductive RQS refinement* process deduces quality requirements on the basis of the type of industrial process that is concerned and the purpose of the environmental inventory (e.g., to report about annual emissions);
- the *current RQS maintenance* process keeps track of the contents of the current set of quality requirements.

The sub-processes of an RQS modification process as distinguished in Chapter 9 were interpreted as follows:

- the *RQS modification analysis* process analyses the results of modifying the current set of quality requirements, to check whether the last modification has resulted in a set of requirements imposing a level of quality that is as least as high as government standards dictate;
- the *RQS modification determination* process determines initial or new quality requirements.

Design object description manipulation

The design object description manipulation process of an environmental inventory process attempts to generate a satisfactory process description, consisting of a process model of the concerned industrial process and information about the environmental impact of this process. The model of the industrial process is determined by asking the user (e.g., a company staff member) for production information such as annual turn-over, the type of raw material and the type of production machinery, and by using defaults for missing pieces of production information. In the context of environmental inventory, the sub-processes of a design object description manipulation process as distinguished in Chapter 8 were interpreted as follows:

- the *DOD modification* process determines an initial process model, derives environmental impact information from the process model, determines the quality of the derived environmental impact information, checks whether this information violates the given quality requirements, and replaces low-quality production information within the process model by information obtained from more reliable sources or by means of more reliable methods, until the environmental impact information that can be derived from the process model satisfies the given quality requirements;
- the *DOD modification history maintenance* process records the history and rationale of proposing, implementing and undoing modifications to process descriptions;
- the *deductive DOD refinement* process deduces implicit production information and environmental impact information from a (partial) process model;
- the *current DOD maintenance* process keeps track of the contents of the current process description.

During DOD modification, defaults about a specific industrial process are established or replaced by facts. The extent to which facts are needed depends on the quality requirements. For example, for an official permit to use a chimney as an exhaust for steam, production-related information has to be gathered to calculate the required peak emissions; but for an annual emission report about a specific production line, a simple emission measurement at the end of the line may suffice.

The sub-processes of a DOD modification process as distinguished in Chapter 9 were interpreted as follows:

- the *DOD modification analysis* process analyses the results of modifying the current process description. It determines whether the last modification has resulted in a process description of which the environmental impact information does not violate any quality requirements;
- the *DOD modification determination* process determines initial or revised values of object attributes.

These processes are composed of more fine-grained processes, such as a process to determine defaults for missing production information; for details, see [Langen, Brazier, Pulles and Diepenmaat, 1995].

11.1.1.2 Process abstraction levels

Being a design process, the upper levels of process abstraction of an environmental inventory process are the same as distinguished in GDM and its specialisations described in Chapter 9. For details of the lower levels, see [Langen, Brazier, Diepenmaat and Pulles, 1995].

11.1.2 Composition of processes

This section describes the way in which the higher-level processes involved in an environmental inventory process are composed of lower-level processes, in terms of possibilities for exchange of information between processes and in terms of task control knowledge used to control both the processes and the information exchange.

11.1.2.1 Information exchange

Being a design process, the exchange of information between the upper-level processes involved in an environmental inventory process is the same as distinguished in GDM and its specialisations described in Chapter 9. For details about the exchange of information between the lower-level processes, see [Langen, Brazier, Diepenmaat and Pulles, 1995].

11.1.2.2 Task control

Being a design process, the knowledge to control the upper-level processes involved in an environmental inventory process is the same as distinguished in GDM and its specialisations described in Chapter 9. For details about the knowledge to control the lower-level processes, see [Langen, Brazier, Diepenmaat and Pulles, 1995].

11.2 Knowledge Composition

This section describes the knowledge structures involved in an environmental inventory process at different levels of reflection. First the levels of reflection are described and then the structure and composition of the knowledge at each of these reflection levels.

11.2.1 Reflection levels

In the context of a Dutch environmental inventory process, the four reflection levels distinguished in GDM are interpreted as follows:

- the *object level* includes information about object attribute values that together form a description of an industrial process. This level also includes knowledge about relations between attribute values of objects and about relations between objects.

- the *first meta-level* includes information about quality requirements and about process descriptions. This level also includes deductive refinement knowledge to deduce standard quality requirements and defaults for object attribute values.

- the *second meta-level* includes information about sets of quality requirements. This level also includes strategic knowledge to determine modifications of quality requirements and to determine modifications of process descriptions, respectively.

- the *third meta-level* includes strategic knowledge to determine a fixed overall strategy for environmental inventory.

11.2.2 Knowledge structures and composition

This section describes the composition and the structure of the knowledge related to environmental inventory and shows how they are modelled in DESIRE.

11.2.2.1 Structure and composition of knowledge at the object level

The object level includes information and knowledge about objects, attributes and values.

Information types

The main organisational structure used is an object-oriented model, which conceptualises industrial processes (e.g., a brick fabrication process and a drying process), their attributes (e.g., the annual turn-over) and part-of relations between these processes (e.g., between a drying process and a brick fabrication process). Figure 11.2 depicts part of the object-oriented model for environmental inventory.

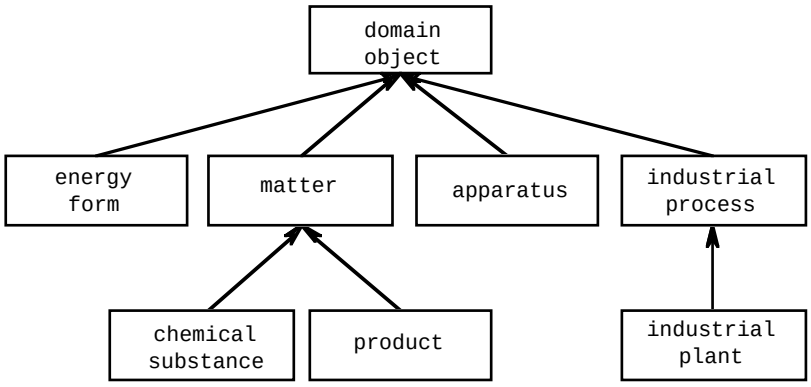


FIGURE 11.2. Part of the object-oriented model for environmental inventory (arrows denote is-a relations).

The second type of structure used is an object-attribute-value structure. A product, for example, has among others attributes for its name, for the unit in which the production of that product is measured and for the (yearly) quantity of the production of that product. Figure 11.3 shows part of the object-attribute-value structure for a product.

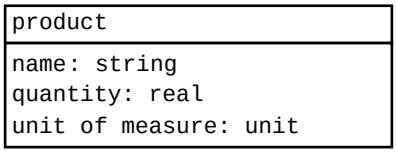


FIGURE 11.3. Part of the object-attribute-value structure for a product.

In the environmental inventory ontology, each object attribute is defined to have a value of a specific type. The following basic types of value are distinguished:

- *integer* for values of attributes such as the number of ovens used to run a brick-baking process in brick fabrication;

- *string* for values of attributes such as the name of a specific industrial company that is the subject of the environmental inventory;
- *integer-unit volume* for values of attributes such as the use of natural gas by a drying process in brick fabrication (e.g., 15 m^3 per 1000 bricks);
- *real-unit volume* for values of attributes such as the emission of carbon dioxide by a drying process in brick fabrication (e.g., 1.3 g per m^3 of used natural gas);
- *domain object list* for values of attributes such as the apparatus used to run a drying process in brick fabrication (e.g., a dryer) or the types of emissions of a process.

Figure 11.4 shows a value type hierarchy with the generic type value at the top and with sub-types that model the values of domain-specific object attributes.

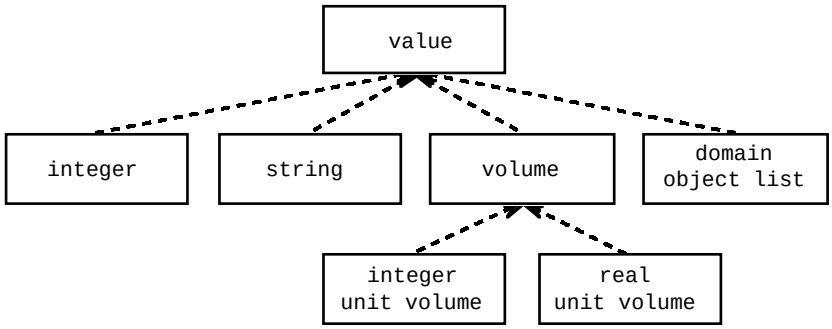


FIGURE 11.4. Value type hierarchy (arrows denote is-a relations).

To refine GDM for the environmental impact inventory domain, GDM's information type domain-object-type was instantiated with objects for different industrial processes, apparatus, matters, and energy forms. (Note that only a fragment of the full model is shown.)

information type domain-object-type
 information types
  industrial-process-type, apparatus-type, matter-type, energy-form-type;

 sorts
  domain-object

 subsorts
  industrial-process, apparatus, matter, energy-form: domain-object;
end information type

information type matter-type
information types
substance-type, product-type;

sorts
matter

subsorts
substance, product: matter;
end information type

In addition, GDM's information type value-type was extended with sub-sorts for integers, strings, volumes and domain object lists.

information type value-type
information types
integer-type, string-type, volume-type, domain-object-list-type;

sorts
value

subsorts
integer, string, volume, domain-object-list: value;
end information type

The is-a relations between the different types of volume defined by the value type hierarchy were modelled by sub-sort relations.

information type volume-type
information types
integer-unit-volume-type, real-unit-volume-type;

sorts
volume

subsorts
integer-unit-volume, real-unit-volume: volume;
end information type

Another important aspect to model of an environmental inventory process is the *quality* of a piece of environmental impact information. The quality denotes how reliable, credible or

representative that piece of information is. In an environmental inventory process, the quality of a specific piece of environmental impact information depends on the quality of the information sources consulted for obtaining production information as well as the methods used to obtain this information.

There are several ways to establish a measure of quality. In a procedural approach, the quality of a specific piece of information would be defined in terms of the procedure followed to produce this information: if the right methods are applied at the right time and in the right order, then the quality of the information is assumed to be acceptable. A more declarative approach would be to rate the information source on which the information is based: a higher rating implies a higher quality of the information.

To instantiate GDM to the environmental inventory domain, GDM's information type application-specific-domain-object-information was extended with references to the application specific information types domain-object-information-sources, domain-object-information-quality-levels and domain-object-information-category-relations.

information type application-specific-domain-object-information
  information types
    domain-object-information-sources,
    domain-object-information-quality-levels,
    domain-object-information-categories;
end information type

To model the quality of an object attribute value, a simple declarative approach has been used, which assumes defaults, facts and derivations to be the only information sources. Information about the source of an object attribute value was modelled by a relation within the information type domain-object-information-sources.

information type source-type
  sorts
    source

  objects
    default, derivation, fact: source;
end information type

information type domain-object-information-sources
  information types
    domain-object-type, attribute-type, source-type;

relations
has-source: domain-object * attribute * source;
end information type

Furthermore, the possible assessment results of the quality of domain object information are assumed to be totally ordered. Information about the quality was modelled by a relation within the information type domain-object-information-quality-levels.

information type domain-object-information-quality-levels
information types
domain-object-type, attribute-type, real-type;

relations
has-quality: domain-object * attribute * real;
end information type

The kind of environmental impact report to be written determines which kind of environmental impact information is to be produced. The different kinds of information correspond to specific types of attribute of an industrial process. To model this correspondence, GDM's generic sort attribute was extended with domain-specific objects for the different kinds of environmental impact information.

information type attribute-type
information types
matter-type, apparatus-type, energy-form-type;

sorts
end-product-information, process-composition-information, product-information,
raw-material-information, apparatus-information, energy-use-information,
energy-production-information, emission-information, waste-information, attribute

subsorts
end-product-information, process-composition-information, product-information,
raw-material-information, apparatus-information, energy-use-information,
energy-production-information, emission-information, waste-information: attribute;

objects
produced-end-products: end-product-information;
total-mass-of-raw-materials: raw-material-information;
total-mass-of-products: product-information;
total-amount-of-used-energy: energy-use-information;

```
    total-amount-of-produced-energy: energy-production-information;
    total-mass-of-emissions: emission-information;
    total-mass-of-waste: waste-information;

  functions
    end-product-volume: matter -> end-product-information;
    sub-processes: matter -> process-composition-information;

    produced-products: matter -> product-information;
    product-volume: matter * matter -> product-information;
    total-product-volume: matter -> product-information;

    used-raw-materials: matter * matter -> raw-material-information;
    raw-material-volume: matter * matter * matter -> raw-material-information;
    total-raw-material-volume: matter -> raw-material-information;

    used-apparatus: matter * matter -> apparatus-information;
    apparatus-quantity: matter * matter * apparatus -> apparatus-information;

    used-energy: matter * matter * apparatus -> energy-use-information;
    energy-use-volume: matter * matter * apparatus * energy-form -> energy-use-information;
    total-energy-use-volume: energy-form -> energy-use-information;

    produced-energy: matter * matter * apparatus * energy-form -> energy-production-information;
    energy-production-volume:
      matter * matter * apparatus * energy-form * energy-form -> energy-production-information;
    total-energy-production-volume: energy-form -> energy-production-information;

    emission: matter * matter * apparatus * energy-form -> emission-information;
    emission-volume: matter * matter * apparatus * energy-form * matter -> emission-information;
    total-emission-volume: matter → emission-information;

    waste: matter * matter * apparatus * energy-form -> waste-information;
    waste-volume: matter * matter * apparatus * energy-form * matter -> waste-information;
    total-waste-volume: matter -> waste-information;
  end information type
```

Knowledge bases

This section presents parts of the knowledge base for the process of deductive DOD refinement, which consists of a knowledge base for value determination and a knowledge base for quality determination.

Value determination. The values of some attributes of an industrial process can be deduced from those that are already known. For example, non-nuclear industrial processes obey the following conservation laws:

$$m_{\text{products}} = m_{\text{raw material}} - m_{\text{emissions}} - m_{\text{waste}} \quad (11.1)$$

$$E_{\text{production}} = E_{\text{use}} \quad (11.2)$$

where m denotes mass and E denotes energy. The first law states that the mass of products produced by an industrial process is equal to the mass of the raw materials used by that process minus the sum of the mass of the matter emitted by that process and the mass of the waste by that process. The second law states that the amount of energy produced by an industrial process is equal to the amount of energy used by that process.

These conservation laws were modelled by rules within the knowledge base of the component deductive-DOD-refinement. For the sake of brevity, only the rules for determining the total mass of products of an industrial process and the total amount of energy produced by an industrial process are shown below.

```
if has-value(P: industrial-process, total-mass-of-raw-materials, val(V1: real, U: unit))
  and has-value(P: industrial-process, total-mass-of-emissions, val(V2: real, U: unit))
  and has-value(P: industrial-process, total-mass-of-waste, val(V3: real, U: unit))
  and V4: real = V1: real – (V2: real + V3: real)
then has-value(P: industrial-process, total-mass-of-products, val(V4: real, U: unit));

if has-value(P: industrial-process, total-amount-of-used-energy, val(V: real, U: unit))
then has-value(P: industrial-process, total-amount-of-produced-energy, val(V: real, U: unit));
```

Industrial processes may be part of a production line. A production line is a composed process with two or more sub-processes. On the basis of the part-of relation between industrial processes, the emissions by a composed process are calculated from the emissions by its sub-processes as follows. If N sub-processes of a composed process P emit a specific matter (e.g., carbon dioxide) in the respective quantities $x_1, x_2, \ldots, x_N$, then the total emission of that matter by P equals $x_1 + x_2 + \ldots + x_N$. This knowledge was modelled by the following rules within the domain-specific knowledge base of the component deductive-DOD-refinement:

```
has-total-volume([ ], total-emission-volume(M: matter), val(0, U: unit));

if has-value(P: industrial-process, total-emission-volume(M: matter), val(V1: real, U: unit))
  and has-total-volume(
    L: industrial-process-list, total-emission-volume(M: matter), val(V2: real, U: unit))
  and V3: real = V1: real + V2: real
```

```
then has-total-volume([P: industrial-process | L: industrial-process-list],
  total-emission-volume(M: matter), val(V3: real, U: unit));

if has-value(P1: industrial-process, sub-processes(P2: product), L: industrial-process-list)
  and has-total-volume(L: industrial-process-list, total-emission-volume(M: matter), V: volume)
then has-value(P1: industrial-process, total-emission-volume(M: matter), V: volume);
```

The calculations of the volumes of raw material, products and waste, as well as energy use and energy production by a composed process, were modelled in a similar way.

Quality determination. For a specific domain object information element E (e.g., the emission volume of a specific matter by a specific process), let N be the number of object attribute values used to determine E and let $0 \leq D \leq N$ be the number of defaults used in the determination of E. Then a quantitative, normalised measure Q_E for the quality of E is the following:

$$Q_E = (N - D) / N \tag{11.3}$$

Hence, $Q_E = 0$ if only defaults were used and $Q_E = 1$ if only facts were used. Thus, the quality of a specific value assignment is the average of the quality of those value assignments used as a basis for the determination of that value. This knowledge was modelled by the following facts and rules within the domain-specific knowledge base of the component deductive-DOD-refinement (for the sake of brevity, only the determination of the quality of the information about the emission volume of a specific matter by a process has been elaborated):

```
if has-source(O: domain-object, A: attribute, default)
then has-quality(O: domain-object, A: attribute, 0);

if has-source(O: domain-object, A: attribute, fact)
then has-quality(O: domain-object, A: attribute, 1);

has-length([ ], 0);

if has-length(L: value-list, N1: integer)
  and N2: integer = N1: integer +1
then has-length([V: value | L: value-list], N2: integer);

has-total-quality([ ], total-emission-volume(M: matter), 0);

if has-quality(P: industrial-process, total-emission-volume(M: matter), Q1: real)
  and has-total-quality(L: industrial-process-list, total-emission-volume(M: matter), Q2: real)
  and has-length(L: industrial-process-list, N: integer)
```

```
  and Q3: real = (Q1: real + N: integer * Q2: real) / (N: integer + 1)
then has-total-quality([P: industrial-process | L: industrial-process-list],
  total-emission-volume(M: matter), Q3: real);

if has-quality(P1: industrial-process, sub-processes(P2: product), Q1: real)
  and has-total-quality(L: industrial-process-list, total-emission-volume(M: matter), Q2: real)
  and has-length(L: industrial-process-list, N: integer)
  and Q3: real = (Q1: real + N: integer * Q2: real) / (N: integer + 1)
then has-quality(P1: industrial-process, total-emission-volume(M: matter), Q3: real);
```

11.2.2.2 Structure and composition of knowledge at the first meta-level

The first meta-level includes information and knowledge about process descriptions, quality requirements and default values of object attributes.

Information types

The environmental inventory ontology contains concepts for process descriptions and quality requirements.

Process descriptions. In an environmental inventory process a *current description* is distinguished, which is the most recent process description that resulted from accepted modifications. By defining the sort process-description as a sub-sort of the generic sort DOD-name, GDM's information type DOD-name-type was instantiated to the environmental inventory domain.

```
information type DOD-name-type
  information types
    process-description-type;

  sorts
    DOD-name

  subsorts
    process-description: DOD-name;
end information type
```

The current process description was modelled as an object of the sort process-description.

```
information type process-description-type
  sorts
    process-description
```

objects
current-description: process-description;
end information type

Quality requirements. There are different kinds of environmental impact report that may have to be written: an annual emission report, an environmental balance sheet, an environmental permit or an environmental prospect report. The type of environmental impact report determines which types of environmental impact information need to be supplied: information about the use of raw materials or energy, or about the products, emissions, waste or energy produced (see Figure 11.1 at the beginning of this chapter). It also determines the level of quality of the environmental impact information to be supplied.

By including references to the information types report-writing-information, process-and-environmental-impact-information-relations and required-environmental-impact-information-quality-levels, GDM's information type application-specific-design-requirement-information was instantiated to the environmental inventory domain.

information type application-specific-design-requirement-information
information types
report-writing-information, process-and-environmental-impact-information-relations,
required-environmental-impact-information-quality-levels;
end information type

The different types of environmental impact reports that may have to be written about an industrial process were modelled as sorts. The is-a relations between these different types were modelled by sub-sort relations.

information type environmental-impact-report-type
sorts
emission-year-report, environmental-permit,
environmental-balance-sheet, environmental-prospectus, environmental-impact-report

objects
EmissionYearReport1, …: emission-year-report;
EnvironmentalPermit1, …: environmental-permit;
EnvironmentalBalanceSheet1, …: environmental-balance-sheet;
EnvironmentalProspectus1, …: environmental-prospectus;

subsorts
emission-year-report, environmental-permit,
environmental-balance-sheet, environmental-prospectus: environmental-impact-report;
end information type

The different kinds of environmental impact information that may have to be supplied in an environmental impact report about an industrial process were modelled as objects of the sort kind-of-information.

information type kind-of-information-type
sorts
kind-of-information

objects
end-product-information, process-composition-information, product-information,
raw-material-information, apparatus-information, energy-use-information,
energy-production-information, emission-information, waste-information:
kind-of-information;
end information type

Information about the kind of environmental impact report that is to be written about a specific industrial process was modelled by a relation within the information type report-writing-information.

information type report-writing-information
information types
environmental-impact-report-type, industrial-process-type;

relations
is-to-be-written-about: environmental-impact-report * industrial-process;
end information type

Information about the kind of environmental impact information that a specific attribute of an industrial process represents was modelled by a relation within the information type process-and-environmental-impact-information-relations.

information type process-and-environmental-impact-information-relations
information types
industrial-process-type, attribute-type, kind-of-information-type;

relations
represents-environmental-impact-information-of-type:
industrial-process * attribute * kind-of-information;
end information type

Information about the level of quality required of a specific kind of environmental impact information that is part of a specific environmental impact report was modelled by a relation within the information type required-environmental-impact-information-quality-levels.

```
information type required-environmental-impact-information-quality-levels
  information types
    environmental-impact-report-type, kind-of-information-type, real-type;

  relations
    requires-environmental-impact-information-with-minimum-quality:
      environmental-impact-report * kind-of-information * real;
end information type
```

Standard names of requirements and qualified requirements for each possible combination of environmental impact report, industrial process and kind of environmental impact information were modelled by name assignment functions within GDM's information types requirement-name-type and qualified-requirement-name-type.

```
information type requirement-name-type
  information types
    environmental-impact-report-type, industrial-process-type, kind-of-information-type;

  sorts
    requirement-name

  functions
    req: environmental-impact-report * industrial-process * kind-of-information ->
      requirement-name;
end information type

information type qualified-requirement-name-type
  information types
    environmental-impact-report-type, industrial-process-type, kind-of-information-type;

  sorts
    requirement-name

  functions
    qualified-req: environmental-impact-report * industrial-process * kind-of-information ->
      qualified-requirement-name;
end information type
```

Knowledge bases

This section presents parts of the knowledge bases for the process of deductive RQS refinement and for the process of determining default domain object information.

Deductive RQS refinement. For the type of industrial process that is concerned, there is knowledge available about the quality requirements to be imposed, given the specific purpose of making an environmental inventory. Using GDM, government standards for the quality of the environmental impact information are modelled by general facts and rules within the knowledge base of the component deductive-RQS-refinement; refer to Section 11.4 for an example. Using GDM, the derivations of quality requirements based on these government standards are modelled by the following general facts and rules within the knowledge base of the component deductive-RQS-refinement:

```
if is-to-be-written-about(R: environmental-impact-report, P: industrial-process)
  and requires-environmental-impact-information-with-minimum-quality(
    R: environmental-impact-report, K: kind-of-information, MinQ: real)
then is-defined-as(
  req(R: environmental-impact-report, P: industrial-process, K: kind-of-information),
  for-all(Attr, for-all(V, for-all(Q,
    l-implies(l-and(has-value(P: industrial-process, Attr, V),
        l-and(represents-environmental-impact-information-of-type(
          P: industrial-process, Attr, K: kind-of-information),
        has-quality(P: industrial-process, Attr, Q))),
      greater-than-or-equal-to(Q, MinQ: real))))));

if is-to-be-written-about(R: environmental-impact-report, P: industrial-process)
  and requires-environmental-impact-information-with-minimum-quality(
    R: environmental-impact-report, K: kind-of-information, MinQ: real)
then is-defined-as(
      qualified-req(R: environmental-impact-report, P: industrial-process, K: kind-of-information),
      every, [ req(R: environmental-impact-report, P: industrial-process, K: kind-of-information) ] );
```

Object attribute default determination. In an environmental inventory process, a DOD modification determination process starts by determining default values for object attributes. For example, with regards to brick fabrication in the Netherlands, the default value of the emission of carbon dioxide by a drying process fuelled by natural gas is 1.3 g per m^3. Using GDM, default values for a specific type of industrial process are modelled by general facts and rules within the domain-specific knowledge base of the component DOD-determination-determination; refer to Section 11.4 for an example.

11.2.2.3 Structure and composition of knowledge at the second meta-level

The second meta-level includes knowledge about sets of quality requirements, about assessments of process descriptions with respect to (sets of) quality requirements, and about environmental inventory process states. (There was no need to introduce domain-specific information types at the second meta-level.)

Knowledge bases

This section presents parts of the knowledge bases for the processes of analysing modifications of sets of quality requirements and analysing and determining modifications of process descriptions.

RQS modification analysis. In an environmental inventory process, an RQS modification analysis process determines which modifications to the quality requirements are acceptable and which ones are not. A modification is acceptable if and only if the new requirement dictates a quality that is greater than or equal to the government standard. This knowledge was modelled by the following rules within the knowledge base of the component RQS-modification-evaluation, which is a sub-component of the component RQS-modification-analysis:

```
if is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-to-be-written-about(R: environmental-impact-report, P: industrial-process), pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    requires-environmental-impact-information-with-minimum-quality(
      R: environmental-impact-report, K: kind-of-information, MinQ: real), pos)
  and is-selected-RQS-alteration(
    addition-of(is-defined-as(Req: requirement-name, has-quality(O: object, A: attribute, Q: real))))
  and includes-design-requirement-information(CurRN: RQS-name,
    represents-environmental-impact-information-of-type(
      P: industrial-process, A: attribute, K: kind-of-information), pos)
  and Q: real ≥ MinQ: real
then is-acceptable-RQS-alteration(
    addition-of(is-defined-as(Req: requirement-name, has-quality(O: object, A: attribute, Q: real))));

if is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-to-be-written-about(R: environmental-impact-report, P: industrial-process), pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    requires-environmental-impact-information-with-minimum-quality(
      R: environmental-impact-report, K: kind-of-information, MinQ: real), pos)
  and is-selected-RQS-alteration(
    addition-of(is-defined-as(Req: requirement-name, has-quality(O: object, A: attribute, Q: real))))
```

```
  and includes-design-requirement-information(CurRN: RQS-name,
    represents-environmental-impact-information-of-type(
          P: industrial-process, A: attribute, K: kind-of-information), pos),
  and Q: real < MinQ: real
then not is-acceptable-RQS-alteration(
    addition-of(is-defined-as(Req: requirement-name, has-quality(O: object, A: attribute, Q: real))));
```

DOD modification analysis. In an environmental inventory process, a DOD modification analysis process determines which modifications to a process description are acceptable and which ones are not. A modification involves changing the value of a specific piece of process information and/or changing the source of that piece of process information. The first type of change does not necessarily affect the quality of the piece of process information, as the same source may have been used to determine the earlier value. The second type of change does not necessarily affect the value of the piece of process information, as a new source may have been used to determine the same value.

A modification that has been applied to the current process description yields a tentative description. The modification is to be rejected if, in the tentative description, a quality requirement is violated that was satisfied directly before the modification. The rationale is that whatever modification is applied, it should never significantly decrease the quality of the environmental impact information. This knowledge was modelled by the following rule within the knowledge base of the component DOD-modification-evaluation, which is a sub-component of the component DOD-modification-analysis:

```
if is-current-RQS(CurRN: RQS-name)
  and is-current-DOD(CurDN: DOD-name)
  and is-selected-DOD-alteration(A: DOD-alteration)
  and is-resulting-DOD(ResDN: DOD-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(R: requirement-name, E2: domain-object-info-expression), pos)
  and satisfies(CurDN: DOD-name, R: requirement-name)
  and violates(ResDN: DOD-name, R: requirement-name)
then not is-acceptable-DOD-alteration(A: DOD-alteration);
```

By making a closed-world assumption over the predicate is-acceptable-DOD-modification-to, it can be determined that a specific design object description modification is acceptable.

DOD modification determination. In an environmental inventory process, a DOD modification determination process at the start of its activity uses default object attribute values. The knowledge involved was modelled by the following rule within the application specific knowledge base of a sub-component of the component DOD-modification-determination:

```
if is-current-status(active)
  and is-previous-status(idle)
  and is-current-DOD(CurDN: DOD-name)
  and holds(has-default-domain-object-information(
     CurDN: DOD-name, E: domain-object-info-element, S: sign), pos)
then is-selected-DOD-alteration(
  addition-of(domain-object-information(E: domain-object-info-element, S: sign)));
```

11.2.2.4 Structure and composition of knowledge at the third meta-level

The third and highest meta-level includes information and knowledge about environmental inventory strategies.

Information types

In an environmental inventory process, the design strategy followed is to first acquire the quality requirements to be satisfied. These quality requirements are determined by the (local) authorities and depend on the purpose of making the environmental inventory (e.g., an environmental impact report or an application for an environmental permit). Then it is tried to determine a satisfactory process description, given the quality requirements. This trial starts by proposing default values for the object attributes of the process description and then by revising values until all quality requirements are satisfied; a revision is acceptable if and only if it resolves the violation of the quality requirement being processed.

By defining the sort environmental-inventory-strategy-name as a sub-sort of GDM's sort design-strategy-name, GDM's information type design-strategy-name-type was instantiated to the environmental inventory domain.

```
information type design-strategy-name-type
  information types
    environmental-inventory-strategy-name-type;

  sorts
    design-strategy-name

  subsorts
    environmental-inventory-strategy-name: design-strategy-name;
end information type
```

The design strategy in an environmental inventory process is to follow a propose-and-revise approach. This concept was modelled by an object of the sort environmental-inventory-strategy-name.

information type environmental-inventory-strategy-name-type
sorts
environmental-inventory-strategy-name

objects
propose-and-revise: environmental-inventory-strategy-name;
end information type

Knowledge bases
The design process co-ordination process of an environmental impact inventory process maintains a fixed overall design strategy, propose-and-revise, which applies to every stage of the environmental impact inventory process and involves that an initial solution is to be proposed and revised until it is satisfactory. This knowledge was modelled by the following fact within the knowledge base of the component DPC:

```
includes-overall-design-strategy(S: design-process-state, propose-and-revise);
```

11.3 Relation between Compositions of Process and Knowledge

The relation between the process composition and the knowledge composition of environmental inventory is the same as for GDM as described in Chapters 6, 7 and 8 and the specialisations of GDM as described in Chapter 9.

11.4 A Sample Domain: Brick and Tile Fabrication

This section presents the results of the development of a model for the environmental inventory of industrial processes in brick and tile fabrication with the aim of writing and monitoring emission year reports. On the basis of this model, a prototype computer system has been designed and implemented, that demonstrates the applicability and usability of the model.

Brick and tile fabrication is a branch of industry, dedicated to the fabrication of mostly bricks and tiles. In the Netherlands, about 60 companies produce about 1.4 billion bricks and 50 million tiles per year. This knowledge about the type and volume of the end-products of a brick and tile factory is modelled by the following general facts and rules within the knowledge base of the component DOD-modification-determination:

```
if is-to-be-written-about(R: environmental-impact-report, P: brick-and-tile-factory)
then has-default-domain-object-information(EmptyDOD,
  has-value(P: brick-and-tile-factory, end-products, [bricks]), pos);
```

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-and-tile-factory, end-product-volume(bricks), val(23000000, piece/year), pos);
```

On average, a Dutch brick and tile factory has one production line for brick fabrication, which can be described as a chain of four sub-processes: mixing, shaping, drying and brick-baking. Sometimes, in a fifth stage, the bricks or tiles are glazed and again brick-baked.

As glazed bricks are produced in small quantities in the Netherlands, a brick-fabrication production line is assumed to only produce non-glazed bricks. This knowledge about the sub-processes of a brick and tile factory is modelled by the following general facts within the knowledge base of the component DOD-modification-determination:

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-and-tile-factory, sub-processes(bricks), [brick-fabrication]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-fabrication, sub-processes(bricks), [mixing, shaping, drying, brick-baking]),
  pos);
```

Table 11.1 shows defaults for the (intermediate) products produced by the four default sub-processes of a brick and tile fabrication process.

TABLE 11.1. Defaults for products produced by sub-processes of brick and tile fabrication.

Process	*End-product*	*Product*	*Quantity*	*Unit*
Mixing	Bricks	Mixed clay	2.298	kg/piece
Shaping	Bricks	Compressed clay	2.400	kg/piece
Drying	Bricks	Green bricks	1.900	kg/piece
Brick-baking	Bricks	Bricks	1.700	kg/piece

This knowledge about the default types and volumes of (intermediate) products produced by sub-processes of a brick and tile fabrication process is modelled by the following general facts within the knowledge base of the component DOD-modification-determination:

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, produced-products(bricks), [mixed-clay]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, product-volume(bricks, mixed-clay), val(2.298, kg/piece)), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: shaping, produced-products(bricks), [compressed-clay]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: shaping, product-volume(bricks, compressed-clay), val(2.400, kg/piece)), pos);
```

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, produced-products(bricks), [green-bricks]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, product-volume(bricks, green-bricks), val(1.900, kg/piece)), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, produced-products(bricks), [bricks]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, product-volume(bricks, bricks), val(1.700, kg/piece)), pos);
```

Table 11.2 shows defaults for the raw materials (matters or intermediate products) used by the four default sub-processes of a brick and tile fabrication process.

TABLE 11.2. Defaults for raw materials used by sub-processes of brick and tile fabrication.

Process	*End-product*	*(Intermediate) Product*	*Raw material*	*Quantity*	*Unit*
Mixing	Bricks	Mixed clay	Wet clay	1.989	kg/piece
Mixing	Bricks	Mixed clay	Water	0.059	kg/piece
Mixing	Bricks	Mixed clay	Additives	0.250	kg/piece
Shaping	Bricks	Compressed clay	Mixed clay	2.298	kg/piece
Shaping	Bricks	Compressed clay	Sand	0.102	kg/piece
Drying	Bricks	Green bricks	Compressed clay	2.400	kg/piece
Brick-baking	Bricks	Bricks	Green bricks	1.900	kg/piece

This knowledge about the default types and volumes of raw materials used by sub-processes of a brick and tile fabrication process is modelled by the following general facts within the knowledge base of the component DOD-modification-determination:

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, used-raw-materials(mixed-clay, bricks), [wet-clay, water, additives]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, raw-material-volume(bricks, mixed-clay, wet-clay), val(1.989, kg/piece)),
  pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, raw-material-volume(bricks, mixed-clay, water), val(0.059, kg/piece)), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, raw-material-volume(bricks, mixed-clay, additives), val(0.250, kg/piece)),
  pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: shaping, used-raw-materials(bricks, compressed-clay), [mixed-clay, sand]), pos);
```

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: shaping, raw-material-volume(bricks, compressed-clay, mixed-clay),
    val(2.298, kg/piece)), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: shaping, raw-material-volume(bricks, compressed-clay, sand),
    val(0.102, kg/piece)), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, used-raw-materials(bricks, green-bricks), [compressed-clay]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, raw-material-volume(bricks, green-bricks, compressed-clay),
    val(2.400, kg/piece)), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, used-raw-materials(bricks, bricks), [green-bricks]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, raw-material-volume(bricks, bricks, green-bricks),
    val(1.900, kg/piece)), pos);
```

Table 11.3 shows defaults for the apparatus running the four default sub-processes of a brick and tile fabrication process.

TABLE 11.3. Defaults for apparatus running the sub-processes of brick and tile fabrication.

Process	*End-product*	*(Intermediate) Product*	*Apparatus*	*Quantity*
Mixing	Bricks	Chunks of mixed clay	Clay mixer	1
Shaping	Bricks	Compressed clay chunks	Form press	1
Drying	Bricks	Green bricks	Drying room	1
Brick-baking	Bricks	Bricks	Tunnel oven	1

This knowledge about the types and quantities of apparatus for running sub-processes of a brick and tile fabrication process is modelled by the following general facts within the knowledge base of the component DOD-modification-determination:

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, used-apparatus(bricks, mixed-clay), [clay-mixer]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, apparatus-quantity(bricks, mixed-clay, clay-mixer), 1), pos);
```

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: shaping, used-apparatus(bricks, compressed-clay), [form-press]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: shaping, apparatus-quantity(bricks, compressed-clay, form-press), 1), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, used-apparatus(bricks, green-bricks), [drying-room]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, apparatus-quantity(bricks, green-bricks, drying-room), 1), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, used-apparatus(bricks, bricks), [tunnel-oven]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, apparatus-quantity(bricks, bricks, tunnel-oven), 1), pos);
```

Table 11.4 shows defaults for the energy forms used by the four default sub-processes of a brick and tile fabrication process. In this table, 'kWh' stands for kilowatt-hour.

TABLE 11.4. Defaults for energy forms used by sub-processes of brick and tile fabrication.

Process	*End-product*	*Product*	*Apparatus*	*Energy*	*Qty*	*Unit*
Mixing	Bricks	Mixed clay	Clay mixer	Electricity	21.9	kWh
Mixing	Bricks	Mixed clay	Clay mixer	Natural gas	10.0	10^{-3} m^3/piece
Shaping	Bricks	Compressed clay	Form press	Electricity	21.9	kWh
Drying	Bricks	Green bricks	Drying room	Electricity	32.8	kWh
Drying	Bricks	Green bricks	Drying room	Natural gas	15.0	10^{-3} m^3/piece
Brick-baking	Bricks	Bricks	Flame oven	Electricity	32.8	kWh
Brick-baking	Bricks	Bricks	Flame oven	Natural gas	140.0	10^{-3} m^3/piece
Brick-baking	Bricks	Bricks	Tunnel oven	Electricity	32.8	kWh
Brick-baking	Bricks	Bricks	Tunnel oven	Natural gas	110.0	10^{-3} m^3/piece

This knowledge about the types and volumes of energy forms used by sub-processes of a brick and tile fabrication process is modelled by the following general facts within the knowledge base of the component DOD-modification-determination:

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, used-energy(bricks, mixed-clay, clay-mixer), [electricity, natural-gas]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, energy-use-volume(bricks, mixed-clay, clay-mixer, electricity),
    val(21.9, kWh)),
  pos);
```

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: mixing, energy-use-volume(bricks, mixed-clay, clay-mixer, natural-gas),
    val(10.0, 10^(–3)*m^3/piece)), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: shaping, used-energy(bricks, compressed-clay, form-press), [electricity]), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: shaping, energy-use-volume(bricks, compressed-clay, form-press, electricity),
    val(21.9, kWh)), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, used-energy(bricks, green-bricks, drying-room), [electricity, natural-gas]),
  pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, energy-use-volume(bricks, green-bricks, drying-room, electricity),
    val(32.8, kWh)), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, energy-use-volume(bricks, green-bricks, drying-room, natural-gas),
    val(15.0, 10^(–3)*m^3/piece)), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, used-energy(bricks, bricks, flame-oven), [electricity, natural-gas]),
  pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, energy-use-volume(bricks, bricks, flame-oven, electricity),
    val(32.8, kWh)), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, energy-use-volume(bricks, bricks, flame-oven, natural-gas),
    val(140.0, 10^(–3)*m^3/piece)), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, used-energy(bricks, bricks, tunnel-oven), [electricity, natural-gas]),
  pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, energy-use-volume(bricks, bricks, tunnel-oven, electricity),
    val(32.8, kWh)), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: brick-baking, energy-use-volume(bricks, bricks, tunnel-oven, natural-gas),
    val(110.0, 10^(–3)*m^3/piece)), pos);
```

The drying process and the brick-baking process of a brick and tile fabrication process are solely responsible for the emissions produced. Table 11.4 shows defaults for the emissions produced by a drying process. In this table, 'VOC' stands for volatile organic compounds, 'No_x' for nitrogen dioxide and nitrogen trioxide, 'CO' for carbon monoxide, 'CO_2' for carbon dioxide and 'H_2O (g)' for water vapour. The emission of H_2O (g) is independent of the form of energy.

TABLE 11.4. Defaults for emissions produced by a drying process.

Process	*End-product*	*Product*	*Apparatus*	*Energy*	*Emission*	*Qty*	*Unit*
Drying	Bricks	Green bricks	Drying room	Natural gas	VOC	1.0	10^{-3} kg/m^3
Drying	Bricks	Green bricks	Drying room	Natural gas	NO_x	1.3	10^{-3} kg/m^3
Drying	Bricks	Green bricks	Drying room	Natural gas	CO	1.3	10^{-3} kg/m^3
Drying	Bricks	Green bricks	Drying room	Natural gas	CO_2	1770.0	10^{-3} kg/m^3
Drying	Bricks	Green bricks	Drying room	(any form)	H_2O (g)	0.5	l/piece

This knowledge about the types and volumes of emissions by a brick drying process is modelled by the following general facts within the knowledge base of the component DOD-modification-determination:

```
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, emission(bricks, green-bricks, drying-room, natural-gas),
    [VOC, NOx, CO, CO2, H2O-g]), pos);

has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, emission-volume(bricks, green-bricks, drying-room, natural-gas, VOC),
    val(1.0, 10^(–3)*kg/m^3)), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, emission-volume(bricks, green-bricks, drying-room, natural-gas, NOx),
    val(1.3, 10^(–3)*kg/m^3)), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, emission-volume(bricks, green-bricks, drying-room, natural-gas, CO),
    val(1.3, 10^(–3)*kg/m^3)), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying, emission-volume(bricks, green-bricks, drying-room, natural-gas, CO2),
    val(1770, 10^(–3)*kg/m^3)), pos);
has-default-domain-object-information(N: DOD-name,
  has-value(P: drying,
    emission-volume(bricks, green-bricks, drying-room, F: energy-form, H2O-g),
    val(0.5, l/piece)), pos);
```

Dutch government standards for the quality of the environmental impact information about brick and tile fabrication were not available at the time of writing this chapter. Suppose, as an example, that the quality of the information about energy use and emission within an emission year report must be at least 0.6; that is, at least sixty per cent of the information must be based on facts. This knowledge is modelled by the following general fact within the knowledge base of the component deductive-RQS-refinement:

```
requires-environmental-impact-information-with-minimum-quality(
  emission-year-report, energy-use-information, 0.6);
requires-environmental-impact-information-with-minimum-quality(
  emission-year-report, emission-information, 0.6);
```

11.5 Discussion

To study its applicability to different types of design process, the generic design model GDM has been used to create a model of environmental inventory. In this type of design process, the object to be designed is a model of a specific industrial process, on the basis of which the environmental impact of this process is derived. The design requirements are conditions on the quality of the environmental impact information about this industrial process.

In order to test the usefulness of the model, it has been applied to the domain of brick and tile fabrication and implemented as a software prototype running on PC. The environmental inventory model turned out to be correct and complete in the sense that, except for instantiations, no specialisations or other modifications had to be made to use the model for environmental inventory of brick and tile fabrication. One successful test alone is not enough to draw conclusions about the usefulness of the model in general. Confidence will only grow by putting the model to the test in more domains of application.

This chapter has shown how GDM, in combination with the specialisations presented in Chapter 9, is used in modelling a practical design application. GDM has proven to be useful in modelling a specific type of design process (environmental inventory) in a specific domain of application (brick and tile fabrication in the Netherlands).

Chapter 12

Strategic Knowledge in Design

In interactive design, a designer and a design support system interact about a strategy for design. This chapter argues that to support interactive design processes, a design support system has to be capable of reasoning with strategic knowledge corresponding to the types of strategic interaction that the system may have with the designer. This chapter explains three types of strategic knowledge: knowledge about the design process as a whole, about the manipulation of requirement qualification sets, and about the manipulation of design object descriptions. An example shows how GDM can be used to model strategic reasoning involved in interactive design.

Publications. *This chapter is based on earlier research on the role of strategic knowledge in design processes [Brazier, Langen and Treur, 1998*a*; Brazier, Langen and Treur, 1998*b*].*

Part V of this thesis is dedicated to three design themes that are regular subjects of research in the AI in Design community: strategic reasoning in design (this chapter), design rationale (Chapter 13), and management of conflicts in design (Chapter 14). The part demonstrates the usability of GDM in modelling theme-related aspects of design processes.

This chapter is organised as follows. Section 12.1 explains the role of strategic knowledge in design and introduces strategic interaction about the design process as a whole, about requirement qualification set manipulation, and about design object description manipulation. For each of the three interaction types, Sections 12.2 to 12.4 present examples of strategic knowledge, together with specifications using DESIRE (Chapter 5). The sections show how these specifications are related to the generic design model GDM (Chapters 6 to 8). Finally, Section 12.5 discusses the usability of GDM in modelling strategic knowledge within interactive design and compares the contributions of this chapter with related research.

12.1 The Role of Strategic Knowledge in Design

In an *interactive design process*, a designer and a design support system interact with each other about various aspects of the design process, such as the design requirements to maintain, the method to be used for modification of the current design object description, and the strategy to be used by the design process. An *(overall) design strategy* is a dynamic plan that includes design activities, precedence relations between these design activities, and conditions stating to which situations these design activities apply. An example of a design strategy is propose-and-revise, which commands to propose an initial 'solution' (i.e., a requirement qualification set and a design object description) on the basis of heuristic knowledge and to modify the 'solution' until it is satisfactory.

The term *strategic interaction* refers to the interaction between a designer and a design support system about the strategy to be used. At least three types of strategic interaction can be distinguished in an interactive design process:

- *interaction about the design process as a whole*, to control a design process in accordance with the given design process objectives by means of an overall design strategy;
- *interaction about the manipulation of requirement qualification sets*, to control a requirement qualification set (RQS) manipulation process in accordance with the given overall design strategy by means of a local RQS manipulation control strategy;
- *interaction about the manipulation of design object descriptions*, to control a design object description (DOD) manipulation process in accordance with the given overall design strategy by means of a local DOD manipulation control strategy.

To support strategic interaction in interactive design processes, a design support system has to be capable of reasoning with strategic knowledge corresponding to the types of strategic interaction to be supported. Equipped with such knowledge, the design support system is able to propose strategies and provide critiques on strategies proposed by a designer.

Strategic knowledge is often motivated (or at least supported) by the experience built up in earlier design cases in the same domain of application. For example, an overall design strategy for a specific mechanical engineering domain may be to focus at the start of the design process on the generation of a design object description for the given set of design requirements. This strategy is based on experience that the given set of design requirements is usually well defined and so need not be modified immediately. In a specific architectural domain, an overall design strategy may be to analyse, supplement and revise the given design requirements at the start of the design process before paying attention to generating a satisfactory design object description. This strategy is based on experience that the given set of design requirements is usually ill structured and therefore needs to be modified right away.

For developers of design support systems, the question is how to model the deployment of strategies in design processes. The following sections present examples of specifications of strategic knowledge and strategic reasoning. The examples are based on a practical case of

designing a house in the Netherlands. The sections also show the relation between the example specifications and the generic design model GDM described in Chapters 6 to 8. Section 12.2 presents strategic knowledge for the overall design process, Section 12.3 more specific strategic knowledge for requirement qualification set manipulation, and Section 12.4 more specific strategic knowledge for design object description manipulation.

12.2 Strategic Knowledge for Design Process Co-ordination

In a design process, the process of design process co-ordination uses information about the design process itself: design process objectives (i.e., requirements on the design process) and status information about the overall design process. On the basis of this information, the design process co-ordination process determines an overall design strategy and evaluates the design process with respect to the current overall design strategy and the given design process objectives. In GDM, this process is modelled by the component DPC, which is a subcomponent of the component design.

12.2.1 Strategic reasoning about the overall design process

Strategic reasoning about the overall design process requires an overall design strategy. The following example describes simple strategic knowledge used in strategic reasoning about the overall design process.

Example 12.1. "If the duration of the design process is limited to a maximum, then the overall design strategy reads as follows: if the design process currently has 100 hours or more to go, then follow an explorative approach, otherwise follow a practical approach." ■

The strategic knowledge in Example 12.1 can be modelled by the following rules within the knowledge base of DPC (where, for the sake of simplicity, details about calculations and comparisons of duration are left out, as well as definitions of the sorts, objects, functions, and relations that are not part of GDM):

```
;; If less than 100 hours remain to complete the design process, follow a practical approach.

if is-defined-as(PO: process-objective-name, is-maximum-duration(D1: duration))
  and is-current-overall-design-process-state(S: design-process-state)
  and includes-design-process-results-information(
    S: design-process-state, is-start-time(T1: time-point), pos)
  and includes-design-process-results-information(
    S: design-process-state, is-current-time(T2: time-point), pos)
  and D2: duration = T1: time-point – T2: time-point
```

```
  and D3: duration = D1: duration – D2: duration
  and is-shorter-than(D3: duration, 100:00:00)
then includes-overall-design-strategy(S: design-process-state, practical-approach);

;; If 100 or more hours remain to complete the design process, follow an explorative approach.

if is-defined-as(PO: process-objective-name, is-maximum-duration(D1: duration))
  and is-current-overall-design-process-state(S: design-process-state)
  and includes-design-process-results-information(
    S: design-process-state, is-start-time(T1: time-point), pos)
  and includes-design-process-results-information(
    S: design-process-state, is-current-time(T2: time-point), pos)
  and D2: duration = T1: time-point – T2: time-point
  and D3: duration = D1: duration – D2: duration
  and not is-shorter-than(D3: duration, 100:00:00)
then includes-overall-design-strategy(S: design-process-state, explorative-approach);
```

The input provided to a design process co-ordination process using these rules determines the overall design strategy to be followed. If, for example, one of the design process objectives states that the design process may take at most 200 hours, and 20 hours have been spent so far, then design process co-ordination will conclude to follow an explorative approach.

```
Input information of DPC
is-defined-as(PO1, is-maximum-duration(200:00:00))

Internal information of DPC
is-current-overall-design-process-state(State198)
includes-design-process-results-information(
  State198, is-start-time(time(20/09/2000, 08:43:00)), pos)
includes-design-process-results-information(
  State198, is-current-time(time(21/09/2000, 04:43:00)), pos)

Output information of DPC
is-current-overall-design-process-state(State198)
includes-overall-design-strategy(State198, explorative-approach)
```

If, however, the design process objective states that the design process may take at most 100 hours, and 20 hours have been spent so far, then design process co-ordination concludes to follow a practical approach.

Input information of DPC
is-defined-as(PO1, is-maximum-duration(100:00:00))

Internal information of DPC
is-current-overall-design-process-state(State198)
includes-design-process-results-information(
  State198, is-start-time(time(20/09/2000, 08:43:00)), pos)
includes-design-process-results-information(
  State198, is-current-time(time(21/09/2000, 04:43:00)), pos)

Output information of DPC
is-current-overall-design-process-state(State198)
includes-overall-design-strategy(State198, practical-approach)

12.2.2 Evaluation of the overall design process

When the processes of requirement qualification set manipulation and design object description manipulation within a design process have terminated their activity, the design process co-ordination process evaluates their results on the basis of the current overall design strategy and the given design process objectives. In Example 12.1, evaluation involves knowledge to determine whether the design process has exceeded the maximum duration or not. This knowledge can be modelled by the following rules within the knowledge base of DPC:

if is-defined-as(PO: process-objective, is-maximum-duration(MaxD: duration))
  and is-current-overall-design-process-state(CurDPS: design-process-state)
  and includes-design-process-results-information(
    CurDPS: design-process-state, is-start-time(BeginT: time-point), pos)
  and includes-design-process-results-information(
    CurDPS: design-process-state, is-finish-time(EndT: time-point), pos)
  and not is-longer-than(EndT: time-point – BeginT: time-point, MaxD: duration)
then is-satisfied(PO: process-objective);

if is-defined-as(PO: process-objective, is-maximum-duration(MaxD: duration))
  and is-current-overall-design-process-state(CurDPS: design-process-state)
  and includes-design-process-results-information(
    CurDPS: design-process-state, is-start-time(BeginT: time-point), pos)
  and includes-design-process-results-information(
    CurDPS: design-process-state, is-finish-time(EndT: time-point), pos)
  and is-longer-than(EndT: time-point – BeginT: time-point, MaxD: duration)
then is-violated(PO: process-objective);

12.3 Strategic Knowledge for RQS Manipulation

This section illustrates different types of strategic knowledge involved in strategic reasoning for requirement qualification set manipulation. The examples are an extension of Example 12.1.

12.3.1 Strategic reasoning about RQS manipulation

The overall design strategy of a design process influences to a greater or lesser extent the strategy for (local control of) the requirement qualification set manipulation process involved. The following example is used to show how the reasoning involved can be modelled.

Example 12.2. "The consequence for requirement qualification set manipulation of a practical approach is to ignore the client's initial soft requirements, whereas the consequence of an explorative approach is to re-negotiate conflicting initial design requirements." ■

This strategic knowledge for requirement qualification set manipulation can be modelled by the following facts and rules within an application-specific knowledge base of the component RQS-modification:

```
is-implemented-by-local-control-strategy(
  practical-approach, ignore-initial-soft-client-requirements);
is-implemented-by-local-control-strategy(
  explorative-approach, renegotiate-initial-conflicting-requirements);

is-defined-as(ignore-initial-soft-client-requirements,
  is-set-of-criteria-for-RQS-reduction-by-modification(
    [is-introduced-at(start-time), has-qualification(all-possible), has-source(client)]));
is-defined-as(renegotiate-initial-conflicting-design-requirements,
  is-set-of-criteria-for-RQS-reduction-by-modification(
    [is-introduced-at(start-time), is-part-of-inconsistency]));

if is-current-overall-design-process-state(CurDPS: design-process-state)
  and includes-design-strategy(
    CurDPS: design-process-state, OverallSN: design-strategy-name)
  and is-current-control-process-state(CurMPS: design-process-state)
  and is-implemented-by-local-control-strategy(
    OverallSN: design-strategy-name, LocalSN: design-strategy-name)
then includes-design-strategy(CurMPS: design-process-state, LocalSN: design-strategy-name);
```

For example, if RQSMState20 is the current control process state and the overall design strategy is to follow a practical approach, then the output of strategic reasoning with this strategic knowledge includes:

```
includes-design-strategy(RQSMState20, ignore-initial-soft-client-requirements)
```

whereas if the overall design strategy is to follow an explorative approach, then the output of strategic reasoning with this strategic knowledge includes:

```
includes-design-strategy(RQSMState20, renegotiate-initial-conflicting-design-requirements)
```

To implement a local RQS manipulation control strategy, its definition is reflected downwards from the third meta-level of a design process (which includes strategic knowledge to determine design strategies) to the second meta-level of a design process (which includes strategic knowledge to determine modifications of requirement qualification sets).

12.3.2 Reasoning with a strategy for RQS manipulation

Reasoning with a strategy for requirement qualification set manipulation means that the definition of this strategy is used to determine which modifications to the current requirement qualification set are to be proposed. This is done on the basis of criteria for selection of design requirements to be added to, or deleted from, the current set. Given Example 12.2, the applicable strategic knowledge to propose modifications to the current requirement qualification set can be modelled by the following facts and rules within the domain task specific knowledge base at the second meta-level of the component RQS-modification-determination:

```
is-empty([ ]);
not is-empty([C: criterion | CL: criterion-list]);

if is-set-of-criteria-for-RQS-reduction-by-modification(CL: criterion-list)
  and not is-empty(CL: criterion-list)
  and is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
  and meets-selection-criteria(
    CurRN: RQS-name, QR: qualified-requirement-name, CL: criterion-list)
then is-proposed-RQS-alteration(deletion-of(
      is-to-be-satisfied(QR: qualified-requirement-name)));

meets-selection-criteria(CurRN: RQS-name, QR: qualified-requirement-name, [ ]);
```

```
if meets-selection-criteria(CurRN: RQS-name, QR: qualified-requirement-name, CL: criterion-list)
  and meets-selection-criterion(CurRN: RQS-name, QR: qualified-requirement-name, C: criterion)
then meets-selection-criteria(
  CurRN: RQS-name, QR: qualified-requirement-name, [C: criterion | CL: criterion-list]);

if includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
then meets-selection-criterion(
  CurRN: RQS-name, QR: qualified-requirement-name, has-qualification(Q: qualification));

if is-current-status(active)
  and is-previous-status(idle)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
then meets-selection-criterion(
  CurRN: RQS-name, QR: qualified-requirement-name, is-introduced-at(start-time));

if includes-design-requirement-information(
    CurRN: RQS-name, is-source-of(S: source, QR: qualified-requirement-name), pos)
then meets-selection-criterion(
  CurRN: RQS-name, QR: qualified-requirement-name, has-source(S: source));

if includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR1: qualified-requirement-name, Q1: qualification, L1: requirement-name-list),
    pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR2: qualified-requirement-name, Q2: qualification, L2: requirement-name-list),
    pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    are-pairs-of-conflicting-design-requirements(
      QR1: qualified-requirement-name, R1: requirement-name,
      QR2: qualified-requirement-name, R2: requirement-name),
    pos)
then meets-selection-criterion(
  CurRN: RQS-name, QR1: qualified-requirement-name, is-part-of-conflict);
```

Example 12.3 shows a case, in which strategic knowledge is used to determine appropriate modifications to the current requirement qualification set.

Example 12.3. A house is to be built on a plot with a road to its west. During preliminary design, the house's volume and outer walls and the front door's orientation are considered.

With regards to the volume, the client specifies his/her needs and desires in terms of floor space and cost. In interaction with the architect, this is translated into a requirement of the volume of the house to be between 255 and 265 m^3. (Dutch architects use requirements for cubic meters, rather than square meters, as a basis for their designs.)

Given the location of the house, the client indicates a preference for the front door to face west (to provide easy access for guests). Knowing that the prevailing wind comes from the west, the architect would prefer the front door to face south (to provide protection against outdoor conditions). Thus, with respect to accessibility, the best option is that the front door faces west, but with respect to protection, the best option is that the front door faces south.

The third aspect for which the client provides specifications concerns the outer walls: these are not to be built from synthetic material. As possible materials, brick or wood could be used. The architect takes two criteria into account: durability and aesthetic value. The use of brick scores best on durability, whereas the use of wood scores best on aesthetic value. The architect has a preference for durability above aesthetic value. ■

In Example 12.3, there is information about the design requirements imposed by the environment, the client and the designer. Information about the design requirements imposed by the environment, the client, and the designer is modelled as follows:

```
Design requirements imposed by the environment
is-defined-as(Re01, has-value(connection(house1, road1), orientation, west))
is-defined-as(QRe01, every, [Re01])
is-source-of(environment, QRe01)

Design requirements put forward by the client at the start of the design process
is-defined-as(Rc01,
  for-all(V, l-implies(has-value(house1, volume, val(V, m^3)), l-and(ge(V, 255), le(V, 265)))))
is-defined-as(Rc02, l-not(has-value(outer-walls, material, synthetic-material)))
is-defined-as(Rc03, has-value(front-door, orientation, west))
is-defined-as(QRc01, every, [Rc01])
is-defined-as(QRc02, every, [Rc02])
is-defined-as(QRc03, all-possible, [Rc03])
is-source-of(client, QRc01)
is-source-of(client, QRc02)
is-source-of(client, QRc03)

Design requirements put forward by the designer during the design process
is-defined-as(Rd04, has-value(front-door, orientation, south))
is-defined-as(Rd05a, has-value(outer-walls, material, brick))
```

```
is-defined-as(Rd05b, has-value(outer-walls, material, wood))
is-defined-as(QRd04, all-possible, [Rd04])
is-defined-as(QRd05, any, [Rd05a, Rd05b])
is-source-of(designer, QRd04)
is-source-of(designer, QRd05)
```

Information about the design criteria and preferences put forward by both the client and the designer is modelled as follows:

```
applies-criterion-to(client, accessibility, value-of(front-door, orientation))
applies-criterion-to(designer, protection, value-of(front-door, orientation))
applies-criterion-to(designer, durability, value-of(outer-walls, material))
applies-criterion-to(designer, aesthetic-value, value-of(outer-walls, material))
favours-as-value-for(accessibility, west, value-of(front-door, orientation))
favours-as-value-for(protection, south, value-of(front-door, orientation))
favours-as-value-for(durability, brick, value-of(outer-walls, material))
favours-as-value-for(aesthetic-value, wood, value-of(outer-walls, material))
uses-decision-criteria-preference-ordering-for(
  designer, [durability, aesthetic-value], value-of(outer-walls, material))
```

Reasoning according to a practical approach. A practical approach means that the client's initial soft requirements are to be ignored. In the context of Example 12.3, this means that the preference for the front door to face west (see design requirement QRc03) is deleted from the current requirement qualification set. As a consequence, the preference for front door to face south remains (see design requirement QRd04), as well as a choice concerning the requirement for the material for the outer walls: Rd05a demands brick, whereas Rd05b demands wood (see design requirement QRd05).

The designer has two criteria to decide which requirement to choose for the type of material for the outer walls: durability and aesthetic value. According to the criterion of durability, the best requirement is Rd05a, whereas according to the criterion of aesthetic value, the best requirement is Rd05b. Since the designer prefers durability to aesthetic value for the choice of a requirement for the material of the outer walls, it is decided to substitute qualified requirement QRd05 by a new qualified requirement QRd05a, which states that requirement Rd05a must be satisfied.

Given Example 12.3, the knowledge to determine which design requirements within the current requirement qualification set have to be imposed on the design object is composed of two types of knowledge: (1) knowledge to determine whether there are requirements that disagree about the value of a domain object, and (2) knowledge to determine, in case of a disagreement between requirements, which requirement is to be imposed according to which decision criterion. This deductive refinement knowledge can be modelled by the following

facts and rules within the domain task specific knowledge base of the component deductive-RQS-refinement (assuming a standard set membership semantics for the relation is-member-of):

```
;; Two requirements may disagree on the value of a particular domain object attribute.

if is-defined-as( QR: qualified-requirement-name, Q: qualification, L: requirement-name-list)
  and is-member-of(R1: requirement-name, L: requirement-name-list)
  and is-member-of(R2: requirement-name, L: requirement-name-list)
  and is-defined-as(R1: requirement-name, has-value(O: domain-object, A: attribute, V1: value))
  and is-defined-as(R2: requirement-name, has-value(O: domain-object, A: attribute, V2: value))
  and not V1: value = V2: value
then disagree-on(
    R1: requirement-name, R2: requirement-name, value-of(O: domain-object, A: attribute));

if is-defined-as(QR1: qualified-requirement-name, Q1: qualification, L1: requirement-name-list)
  and is-defined-as(
    QR2: qualified-requirement-name, Q2: qualification, L2: requirement-name-list)
  and is-member-of(R1: requirement-name, L1: requirement-name-list)
  and is-member-of(R2: requirement-name, L2: requirement-name-list)
  and is-defined-as(R1: requirement-name, has-value(O: domain-object, A: attribute, V1: value))
  and is-defined-as(R2: requirement-name, has-value(O: domain-object, A: attribute, V2: value))
  and not V1: value = V2: value
then disagree-on(
    R1: requirement-name, R2: requirement-name, value-of(O: domain-object, A: attribute));

if is-defined-as(QR1: qualified-requirement-name, Q1: qualification, L1: requirement-name-list)
  and is-defined-as(
    QR2: qualified-requirement-name, Q2: qualification, L2: requirement-name-list)
  and is-member-of(R1: requirement-name, L1: requirement-name-list)
  and is-member-of(R2: requirement-name, L2: requirement-name-list)
  and is-defined-as(R1: requirement-name, has-value(O: domain-object, A: attribute, V1: value))
  and is-defined-as(R2: requirement-name, has-value(O: domain-object, A: attribute, V2: value))
  and not V1: value = V2: value
then are-pairs-of-conflicting-design-requirements(
    QR1: qualified-requirement-name, R1: requirement-name,
    QR2: qualified-requirement-name, R2: requirement-name);

;; If there is a criterion that can be applied to choose a value for a specific object attribute, then
;; the value favoured by that criterion determines which requirement defined on that same object
;; attribute is to be imposed on the design object.
```

```
if applies-criterion-to(DA: design-agent, C: criterion, value-of(O: domain-object, A: attribute))
  and is-defined-as(R: requirement-name, has-value(O: domain-object, A: attribute, V: value))
  and favours-as-value-for(C: criterion, V: value, value-of(O: domain-object, A: attribute))
then is-to-be-imposed-according-to(R: requirement-name, C: criterion);

if applies-criterion-to(DA: design-agent, C: criterion, value-of(O: domain-object, A: attribute))
  and is-defined-as(R: requirement-name, has-value(O: domain-object, A: attribute, V1: value))
  and favours-as-value-for(C: criterion, V2: value, value-of(O: domain-object, A: attribute))
  and not V1: value = V2: value
then not is-to-be-imposed-according-to(R: requirement-name, C: criterion);

;; If there are two criteria that can be applied to choose a value for the specific object attribute,
;; then these criteria agree to have a specific requirement as a favourite if and only if according
;; to both criteria, that same requirement is to be imposed.

if applies-criterion-to(A1: design-agent, C1: criterion, value-of(O: domain-object, A: attribute))
  and applies-criterion-to(A2: design-agent, C2: criterion, value-of(O: domain-object, A: attribute))
  and is-defined-as(R: requirement-name, has-value(O: domain-object, A: attribute, V: value))
  and is-to-be-imposed-according-to(R: requirement-name, C1: criterion)
  and is-to-be-imposed-according-to(R: requirement-name, C2: criterion)
then agree-to-have-as-favourite(C1: criterion, C2: criterion, R: requirement-name);

if applies-criterion-to(A1: design-agent, C1: criterion, value-of(O: domain-object, A: attribute))
  and applies-criterion-to(A2: design-agent, C2: criterion, value-of(O: domain-object, A: attribute))
  and is-defined-as(R1: requirement-name, has-value(O: domain-object, A: attribute, V1: value))
  and is-defined-as(R2: requirement-name, has-value(O: domain-object, A: attribute, V2: value))
  and not V1: value = V2: value
  and is-to-be-imposed-according-to(R1: requirement-name, C1: criterion)
  and is-to-be-imposed-according-to(R2: requirement-name, C2: criterion)
then not agree-to-have-as-favourite(C1: criterion, C2: criterion, R1: requirement-name);

;; If there are two relevant criteria that disagree to have a specific requirement as a favourite, then
;; that requirement is not an overall favourite of all relevant criteria.

if not agree-to-have-as-favourite(C1: criterion, C2: criterion, R: requirement-name)
then not is-overall-favourite(R: requirement-name);
```

Activation of the component deductive-RQS-refinement yields the following output with regards to the requirements and decision criteria in Example 12.3:

disagree-on(Rc03, Rd04, value-of(front-door, orientation))
disagree-on(Rd04, Rc03, value-of(front-door, orientation))
disagree-on(Rd05a, Rd05b, value-of(outer-walls, material))
disagree-on(Rd05b, Rc05a, value-of(outer-walls, material))

are-pairs-of-conflicting-design-requirements(QRd05, Rd05a, QRd05, Rd05b)
are-pairs-of-conflicting-design-requirements(QRd05, Rd05b, QRd05, Rd05a)

is-to-be-imposed-according-to(Rc03, accessibility)
is-to-be-imposed-according-to(Rd04, protection)
is-to-be-imposed-according-to(Rd05a, durability)
is-to-be-imposed-according-to(Rd05b, aesthetic-value)

not agree-to-have-as-favourite(accessibility, protection, Rc03)
not agree-to-have-as-favourite(protection, accessibility, Rd04)
not agree-to-have-as-favourite(durability, aesthetic-value, Rd05a)
not agree-to-have-as-favourite(aesthetic-value, durability, Rd05b)

not is-overall-favourite(Rc03)
not is-overall-favourite(Rd04)
not is-overall-favourite(Rd05a)
not is-overall-favourite(Rd05b)

Note that it has been assumed that prior to this activation of deductive RQS refinement, the client's initial soft requirements were deleted from the current requirement qualification set, which means that the qualified requirement QRc03 was removed. (A requirement itself need not be deleted, for it is in force only when referred to by at least one qualified requirement.)

After having established that there are conflicting design requirements, criteria are used to determine which of the conflicting design requirements have to be replaced and which ones may remain part of the current requirement qualification set. For each pair of conflicting design requirements ⟨(*QR1, R1*), (*QR2, R2*)⟩, if *R1* is not an overall favourite, then the following modifications are made:

1. *QR1* is selected to be deleted from the current requirement qualification set;
2. a new qualified requirement *QR1'* is generated and selected to be added to the current requirement qualification set, where *QR1'* has the same qualification as *QR1* and its list of requirements is identical to the one referred to by *QR1* except that *R1* is absent.

This knowledge can be modelled by the following facts and rules within an application-specific knowledge base of the component RQS-modification-determination:

```
if is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    are-pairs-of-conflicting-design-requirements(
      QR: qualified-requirement-name, R: requirement-name,
      QRfav: qualified-requirement-name, Rfav: requirement-name), pos)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-overall-favourite(R: requirement-name), neg)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-overall-favourite(Rfav: requirement-name), pos)
then is-proposed-RQS-alteration(deletion-of(
      is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list)));

if is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    are-pairs-of-conflicting-design-requirements(
      QR: qualified-requirement-name, R: requirement-name,
      QRfav: qualified-requirement-name, Rfav: requirement-name), pos)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-overall-favourite(R: requirement-name), neg)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-overall-favourite(Rfav: requirement-name), pos)
  and not L: requirement-name-list = [R: requirement-name]
  and is-difference-of(
    Lprime: requirement-name-list, L: requirement-name-list, [R: requirement-name])
then is-proposed-RQS-alteration(addition-of(
      is-defined-as(prime(QR: qualified-requirement-name, Rfav: requirement-name),
        Q: qualification, Lprime: requirement-name-list)));

;; To model the removal of an item from a list, set difference is used.

is-difference-of([ ], [ ], L: requirement-name-list);

if is-difference-of(
    L3: requirement-name-list, L1: requirement-name-list, L2: requirement-name-list)
  and is-member-of(R: requirement-name, L2: requirement-name-list)
then is-difference-of(L3: requirement-name-list,
    [R: requirement-name | L1: requirement-name-list], L2: requirement-name-list);
```

```
if is-difference-of(
    L3: requirement-name-list, L1: requirement-name-list, L2: requirement-name-list)
  and not is-member-of(R: requirement-name, L2: requirement-name-list)
then is-difference-of([R: requirement-name | L3: requirement-name-list],
    [R: requirement-name | L1: requirement-name-list], L2: requirement-name-list);
```

In Example 12.3, there is no overall favourite requirement for the design requirements in conflict.

To force a ranking of the requirements involved in the conflict, information about preferences between criteria is used. That is, if one criterion is preferred to a second criterion, then the requirement to be imposed according to the first criterion is preferred to the requirement to be imposed according to the second criterion. This knowledge can be modelled by the following facts and rules within the domain task specific knowledge base of the component RQS-modification-determination:

```
occurs-earlier-in-list-than(C1: criterion, [C1: criterion | CL: criterion-list], C2: criterion);

if not C: criterion = C2: criterion
  and occurs-earlier-in-list-than(C1: criterion, CL: criterion-list, C2: criterion)
then occurs-earlier-in-list-than(C1: criterion, [C: criterion | CL: criterion-list], C2: criterion);

if occurs-earlier-in-list-than(C1: criterion, CL: criterion-list, C2: criterion)
then is-more-preferred-than(C1: criterion, C2: criterion, CL: criterion-list);

if is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(R: requirement-name, has-value(O: domain-object, A: attribute, V: value)), pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    are-pairs-of-conflicting-design-requirements(
      QR: qualified-requirement-name, R: requirement-name,
      QR2: qualified-requirement-name, R2: requirement-name), pos)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-overall-favourite(R: requirement-name), neg)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-overall-favourite(R2: requirement-name), neg)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-to-be-imposed-according-to(R: requirement-name, C: criterion), pos)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-to-be-imposed-according-to(R2: requirement-name, C2: criterion), pos)
```

```
  and includes-design-requirement-information(CurRN: RQS-name,
    uses-decision-criteria-preference-ordering-for(
      DA: design-agent, CL: criterion-list, value-of(O: domain-object, A: attribute)), pos)
  and is-more-preferred-than(C2: criterion, C: criterion, CL: criterion-list)
then is-proposed-RQS-alteration(deletion-of(
      is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list)));

if is-current-RQS(CurRN: RQS-name)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(QR: qualified-requirement-name, Q: qualification, L: requirement-name-list), pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    is-defined-as(R: requirement-name, has-value(O: domain-object, A: attribute, V: value)), pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    are-pairs-of-conflicting-design-requirements(
      QR: qualified-requirement-name, R: requirement-name,
      QR2: qualified-requirement-name, R2: requirement-name), pos)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-overall-favourite(R: requirement-name), neg)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-overall-favourite(R2: requirement-name), neg)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-to-be-imposed-according-to(R: requirement-name, C: criterion), pos)
  and includes-design-requirement-information(
    CurRN: RQS-name, is-to-be-imposed-according-to(R2: requirement-name, C2: criterion), pos)
  and includes-design-requirement-information(CurRN: RQS-name,
    uses-decision-criteria-preference-ordering-for(
      DA: design-agent, CL: criterion-list, value-of(O: domain-object, A: attribute)), pos)
  and is-more-preferred-than(C2: criterion, C: criterion, CL: criterion-list)
  and not L: requirement-name-list = [R: requirement-name]
  and is-difference-of(
    Lprime: requirement-name-list, L: requirement-name-list, [R: requirement-name])
then is-proposed-RQS-alteration(addition-of(
      is-defined-as(prime(QR: qualified-requirement-name, R2: requirement-name),
        Q: qualification, Lprime: requirement-name-list)));
```

In Example 12.3, the designer preferred durability as a criterion for the choice of material of the outer wall. Thus, if the component RQS-modification-determination is activated and the resulting proposed modifications are all selected and applied to the current requirement qualification set, the effect is that qualified requirement QRd05 is deleted and a new qualified requirement is added to the current requirement qualification set:

```
is-defined-as(prime(QRd05, Rd05a), any, [Rd05a])
```

In the updated requirement qualification set, there are no conflicting design requirements, so there is no more reason to continue with requirement qualification set manipulation.

Reasoning according to an explorative approach. An explorative approach means that the initial design requirements that are in conflict are to be re-negotiated. In Example 12.3, the qualified requirements regarding the orientation of the front door and the material of the outer walls are in conflict.

In the context of Example 12.3, activation of the component deductive-RQS-refinement yields the following output with regards to conflicting design requirements:

```
disagree-on(Rc03, Rd04, value-of(front-door, orientation))
disagree-on(Rd04, Rc03, value-of(front-door, orientation))
disagree-on(Rd05a, Rd05b, value-of(outer-walls, material))
disagree-on(Rd05b, Rc05a, value-of(outer-walls, material))

are-pairs-of-conflicting-design-requirements(QRc03, Rc03, QRd04, Rd04)
are-pairs-of-conflicting-design-requirements(QRd04, Rd04, QRc03, Rc03)
are-pairs-of-conflicting-design-requirements(QRd05, Rd05a, QRd05, Rd05b)
are-pairs-of-conflicting-design-requirements(QRd05, Rd05b, QRd05, Rd05a)
```

Thus, if the component RQS-modification-determination is activated and the resulting proposed modifications are all selected and applied to the current requirement qualification set, the effect is that the qualified requirements QRc03, QRd04 and QRd05 are deleted. In the updated requirement qualification set, there are no conflicting design requirements, so there is no more reason to continue with requirement qualification set manipulation.

12.4 Strategic Knowledge for DOD Manipulation

This section illustrates different types of strategic knowledge involved in strategic reasoning for design object description manipulation. The examples are an extension of Example 12.1.

12.4.1 Strategic reasoning about DOD manipulation

The overall design strategy of a design process influences to a greater or lesser extent the strategy for (local control of) the design object description manipulation process involved. The following example is used to show how the reasoning involved can be modelled.

Example 12.4. "The consequence for design object description manipulation of a practical approach is to re-use an existing design. The consequence of an explorative approach is to generate a new design from scratch." ■

This strategic knowledge for design object description manipulation can be modelled by the following facts and rules within an application-specific knowledge base at the third meta-level of the component DOD-modification:

```
is-implemented-by-local-control-strategy(practical-approach, reuse-existing-design);
is-implemented-by-local-control-strategy(explorative-approach, generate-design-from-scratch);

is-defined-as(reuse-existing-design, is-set-of-criteria-for-DOD-extension-by-retrieval(
   [satisfies-design-requirement-which([has-source(environment)])]));
is-defined-as(generate-design-from-scratch,
  is-set-of-criteria-for-DOD-extension-by-retrieval([is-empty]));

if  is-current-overall-design-process-state(CurDPS: design-process-state)
  and includes-design-strategy(
    CurDPS: design-process-state, OverallSN: design-strategy-name)
  and is-current-control-process-state(CurMPS: design-process-state)
  and is-implemented-by-local-control-strategy(
    OverallSN: design-strategy-name, LocalSN: design-strategy-name)
then includes-design-strategy(
    CurMPS: design-process-state, LocalSN: design-strategy-name);
```

For example, if DODMState20 is the current control process state and the overall design strategy is to follow a practical approach, then the output of strategic reasoning with this strategic knowledge includes the information that an existing design has to be re-used:

```
includes-design-strategy(DODMState20, reuse-existing-design)
```

whereas if the overall design strategy is to follow an explorative approach, then the output of strategic reasoning includes the information that a design has to be generated from scratch:

```
includes-design-strategy(DODMState20, generate-design-from-scratch)
```

To implement a local DOD manipulation control strategy, its definition is reflected downwards from the third meta-level of a design process (which includes strategic knowledge to determine design strategies) to the second meta-level of a design process (which includes strategic knowledge to determine modifications of design object descriptions).

12.4.2 Reasoning with a strategy for DOD manipulation

Reasoning with a strategy for design object description manipulation means that the definition of this strategy is used to determine which modifications to the current design object description are to be proposed. This is done on the basis of criteria for selection of domain object information to be added to, or deleted from, the current design. Given Example 12.4, the applicable strategic knowledge to propose modifications to the current design object description can be modelled by the following facts and rules within an application-specific knowledge base of the component DOD-modification-determination:

```
is-empty([ ]);
not is-empty([C: criterion | CL: criterion-list]);

if is-set-of-criteria-for-DOD-extension-by-retrieval(CL: criterion-list)
  and not is-empty(CL: criterion-list)
  and is-current-RQS(CurRN: RQS-name)
  and is-current-DOD(CurDN: DOD-name)
  and meets-selection-criteria(CurRN: RQS-name, CurDN: DOD-name, CL: criterion-list)
then is-proposed-DOD-to-be-retrieved(CurDN: DOD-name);

meets-selection-criteria(CurRN: RQS-name, CurDN: DOD-name, [ ]);

if meets-selection-criteria(CurRN: RQS-name, CurDN: DOD-name, CL: criterion-list)
  and meets-selection-criterion(CurRN: RQS-name, CurDN: DOD-name, C: criterion)
then meets-selection-criteria(
  CurRN: RQS-name, CurDN: DOD-name, [C: criterion | CL: criterion-list]);

meets-selection-criterion(CurRN: RQS-name, EmptyDOD, is-empty);

if meets-selection-criteria(CurRN: RQS-name, QR: qualified-requirement-name, CL: criterion-list)
  and includes-basic-evaluation-information(
    CurRN: RQS-name, satisfies(ExistingDN: DOD-name, QR: qualified-requirement-name), pos)
then meets-selection-criterion( CurRN: RQS-name, ExistingDN: DOD-name,
  satisfies-design-requirement-which(CL: criterion-list));
```

(Note that to determine design requirements from a given requirement qualification set that meet given selection criteria, the same knowledge applies as specified for Example 12.3. For the sake of brevity, the facts and rules modelling this knowledge have been left out here.)

In the following, Example 12.3 introduced in Section 12.3.2 shows how strategic knowledge is used to determine appropriate modifications to the current design object description.

Reasoning according to a practical approach. A practical approach means that an existing design is to be re-used. In the context of Example 12.3, this means that designs are to be retrieved from the design history that satisfy the given environment-based design requirements:

```
is-defined-as(Re01, has-value(connection(house1, road1), orientation, west))
is-defined-as(QRe01, every, [Re01])
```

As a result, a number of existing designs are found of a house on a plot with a road to its west. From these designs, one is to be selected as the new current design object description. Which design is chosen depends on the degree to which each design satisfies the remaining design requirements imposed by the environment, the client, and the designer. These design requirements are part of the current requirement qualification set that resulted from following a practical approach.

```
is-defined-as(Rc01,
  for-all(V, l-implies(has-value(house1, volume, val(V, m^3)), l-and(ge(V, 255), le(V, 265)))))
is-defined-as(Rc02, l-not(has-value(outer-walls, material, synthetic-material)))
is-defined-as(Rd04, has-value(front-door, orientation, south))
is-defined-as(Rd05a, has-value(outer-walls, material, brick))
is-defined-as(Rd05b, has-value(outer-walls, material, wood))
is-defined-as(QRc01, every, [Rc01])
is-defined-as(QRc02, every, [Rc02])
is-defined-as(QRd04, all-possible, [Rd04])
is-defined-as(prime(QRd05, Rd05a), any, [Rd05a])
```

Reasoning according to an explorative approach. An explorative approach means that a new design has to be generated from scratch. In the context of Example 12.3, a design has to be generated that satisfies the non-conflicting design requirements imposed by the environment, the client and the designer. These design requirements are part of the current requirement qualification set that resulted from following an explorative approach.

```
is-defined-as(Re01, has-value(connection(house1, road1), orientation, west))
is-defined-as(Rc01,
  for-all(V, l-implies(has-value(house1, volume, val(V, m^3)), l-and(ge(V, 255), le(V, 265)))))
is-defined-as(Rc02, l-not(has-value(outer-walls, material, synthetic-material)))
is-defined-as(Rd04, has-value(front-door, orientation, south))
is-defined-as(Rd05a, has-value(outer-walls, material, brick))
is-defined-as(Rd05b, has-value(outer-walls, material, wood))
is-defined-as(QRe01, every, [Re01])
is-defined-as(QRc01, every, [Rc01])
is-defined-as(QRc02, every, [Rc02])
```

12.5 Discussion

In an interactive design process, a designer and a design support system interact about a strategy that is supposed to lead the design process to a successful end. This chapter has argued that to support interactive design processes, a design support system has to be capable of reasoning with strategic knowledge corresponding to the types of strategic interaction that the system may have with the designer. Three types of strategic knowledge have been distinguished: knowledge about the design process as a whole, about the manipulation of requirement qualification sets, and about the manipulation of design object descriptions.

Using a practical case of designing a house as an example, this chapter has shown how the strategic reasoning involved can be modelled on the basis of GDM. This generic design model has proven to provide a structure for distinguishing and modelling different types of strategic knowledge, on the basis of the functional role that they play in a design process. GDM distinguishes strategic knowledge with respect to the overall design process, requirement qualification set manipulation, and design object description manipulation.

The idea that design strategies, strategic knowledge and strategic reasoning are essential for (interactive) design processes, especially those that generate large design spaces, is widely acknowledged by the AI in Design community. The remainder of this section compares the contributions of this chapter with related research. The role of design strategy, the role of strategic knowledge, and examples of design strategies are discussed.

The role of design strategy. This chapter defines an overall design strategy to be a dynamic plan that includes design activities, precedence relations between these design activities, and conditions stating to which situations these design activities apply. A design strategy may be formulated either on the level of a design process as a whole or on the level of a requirement qualification set manipulation process or a design object description manipulation process. This definition is similar to most of the definitions encountered in AI in Design research, of which some are discussed below.

Akin asserts that designers use knowledge about (physical) objects and concepts as well as strategies to control design processes [Akin, 1978]. A design strategy is a plan to activate specific design mechanisms, among which he distinguishes the following:

- *information acquisition*: acquisition of information about the design problem,
- *information interpretation*: interpretation of information about the design problem,
- *information storage*: storage of information about past actions and their results,
- *partial-solution generation*: generation of a solution that satisfies only one, or a few aspects of the total set of design requirements,
- *solution evaluation*: checking a new partial solution against the criteria, or constraints, used in generating all previous partial solutions,
- *solution integration*: integration of a partial solution into the overall solution,
- *input and output*: visual perception and sketching mechanisms.

Table 12.1 shows how the design mechanisms distinguished by Akin correspond to the design processes distinguished within GDM and its specialisations described in Chapter 9.

TABLE12.1. Design mechanisms by Akin and corresponding design processes in GDM.

Design mechanism	*Corresponding design process*
Information acquisition	RQS modification (all sub-processes)
Information interpretation	Deductive RQS refinement, RQS assessment
Information storage	RQSM history maintenance, DODM history maintenance
Partial-solution generation	DOD modification (all sub-processes)
Solution evaluation	Deductive DOD refinement, DOD assessment
Solution integration	DOD modification (all sub-processes)
Input and output	RQS modification, DOD modification (all sub-processes)

Brown and Chandrasekaran have developed a generic theory of routine design as well as a framework for the compositional analysis of design processes [Brown and Chandrasekaran, 1989]. They model design as a collection of generic tasks, where each generic task is characterised by information about its function (in terms of the types of input and output), representation of the knowledge involved, and an appropriate inference strategy for the function. If a problem matches the function of a generic task, then the generic task provides a knowledge representation as well as an inference strategy that can be used to solve the problem.

Brown and Chandrasekaran state that a design strategy can be expressed as a plan consisting of generic design tasks and their sequential relationships. They distinguish the following generic routine design tasks:

- *decomposition*: decomposition of a given design problem into smaller problems,
- *planning*: planning of design actions to take for achieving a specific design goal or for designing a specific part,
- *matching*: choosing an earlier generated design object description that is “closest” to the current design problem,
- *critiquing*: analysis of the failure of a given design object description to be a solution to the current design problem,
- *modification*: making changes to a design object description, based on an analysis,
- *constraint processing*: satisfaction and propagation of design constraints,
- *goal/constraint generation*: translation of the goals/constraints of a given design problem into goals/constraints of the sub-problems of that design problem,
- *recomposition*: integration of the solutions of the design sub-problems into a solution of the overall design problem,
- *design verification*: testing of the design solution against the original design goals and constraints.

Table 12.2 shows how the generic (routine) design tasks distinguished by Brown and Chandrasekaran correspond to the design processes distinguished within GDM and its specialisations described in Chapter 9.

TABLE12.2. Generic design tasks by Brown and Chandrasekaran and corresponding design processes in GDM.

Generic design task	*Corresponding design process*
Decomposition	RQS modification determination
Planning	DOD modification determination
Matching	DODM history maintenance
Critiquing	DOD modification analysis (in particular DOD assessment)
Modification	DOD modification determination
Constraint processing	Deductive DOD refinement, DOD modification determination
Goal/constraint generation	Deductive RQS refinement, RQS modification determination
Recomposition	DOD modification determination
Design verification	DOD modification analysis (in particular DOD assessment)

To avoid exhaustive search in a huge design space, Treur states that a strategic reasoning layer is needed that contains knowledge about the possible inference steps at the basic reasoning layer, where modifications are generated to design requirement sets and design object descriptions [Treur, 1989]. He defines a design strategy in terms of modifications such as:

- *transformation*: modification of a specification (i.e., a set of design requirements or a design object description) into an equivalent specification in the same language,
- *translation*: modification of a specification into an equivalent specification expressed in a different language,
- *division*: modification of a specification into a set of two complementary specifications, together comprising the original specification,
- *hierarchical decomposition*: modification of a specification into a set of more detailed specifications and a composition relation between these more detailed specifications,
- *reduction*: modification of a specification into a smaller specification (which is part of the original specification),
- *extension*: modification of a specification into a larger specification (of which the original specification is part).

Using the specialisation of GDM described in Chapter 9, these types of modification are modelled as modification methods, except *reduction* and *extension*, which are taken to be the basic kinds of changes made to requirement qualification sets or design object descriptions. (For example, transformation would be modelled as a series of reductions and extensions.)

Strelnikov and Dmitrevich provide a formal description of interactive design strategies accomplished within Computer Aided Design systems [Strelnikov and Dmitrevich, 1991].

They express design strategies as sub-graphs of a graph model of all permissible paths in a design process. Nodes denote different types of specification that play a role within a design process, such as the current design goals, the current design solution and the current knowledge base. Arcs denote different types of operation that can be applied to transform a design specification of one type into another.

Strelnikov and Dmitrevich distinguish the following types of operation within an interactive design system:

- *synthesis*: process control and generation of modifications to the current design description, the database or the knowledge base,
- *analysis*: presentation of the current design description as the current design solution,
- *prediction*: prediction of future process events and design states,
- *comment*: presentation of previous solutions, prompts, help, etc.,
- *explanation*: diagnosis of the current design state and inference of new specifications from the current design state,
- *evaluation*: evaluation of the current design solution and the current design strategy,
- *decision-making*: decision-making with respect to the design goals and constraints to impose.

Table 12.3 shows how the types of operation distinguished by Strelnikov and Dmitrevich correspond to the design processes distinguished within GDM and its specialisations described in Chapter 9.

TABLE 12.3. Types of operation by Strelnikov and Dmitrevich and corresponding design processes in GDM.

Type of operation	*Corresponding design process*
Synthesis	DOD modification determination, DPC
Analysis	DOD modification analysis
Prediction	DOD modification determination
Comment	DOD modification analysis
Explanation	DOD modification analysis
Evaluation	DOD modification analysis, DPC
Decision-making	RQS modification determination, DOD modification determination, DPC

The role of strategic knowledge. This chapter gave examples of strategic knowledge for a design process as a whole, for local control of a requirement qualification set manipulation process, and for local control of a design object description manipulation process. The role of this strategic knowledge has been explained to be twofold: to be able to reason *about* design strategies (with the aim to determine and evaluate design strategies) and also *with* design strategies (with the aim to derive implications of a specific design strategy on modification processes). In AI in Design research, similar views are expressed.

Smithers, Corne and Ross define a control strategy of an explorative design process to determine revisions of requirements descriptions and formulations of well defined problem statements interpreting these requirements descriptions, given the available design knowledge and the design history [Smithers, Corne and Ross, 1994]. The design knowledge consists of knowledge of the domain, knowledge of how to design in this domain, and knowledge of the embedding culture in which the design process is situated (and with which it must successfully interact). The design history records the series of requirements descriptions formed, the well defined problem statements formed as well as the functions by means of which these problem statements have been generated, and the series of design descriptions formed during the explorative design process.

In GDM, knowledge of how to design in a domain is modelled as knowledge structures within design process co-ordination (to determine an overall design strategy), requirement qualification set manipulation (to determine a local control strategy), and design object description manipulation (to determine a local control strategy). Using the specialisation of GDM described in Chapter 9, more detailed knowledge of how to determine revisions of requirements descriptions and formulations of well defined problem statements is modelled as knowledge structures within RQS modification (to determine modification methods).

Hori investigates how design support systems can exploit strategic knowledge [Hori, 1997]. He defines strategic knowledge as the type of knowledge that controls the whole loop of reflection-in-action, where each iteration consists of trying some action, reflecting on the result of the action and determining the next action ([Schön, 1983]). Hori states that applying strategic knowledge is an essential aspect of creative design, as it enables to jump from one conceptual space (defined by a set of variables) to another (by introducing new variables).

In GDM, strategic knowledge as defined by Hori is modelled as knowledge structures within design process co-ordination (to determine and evaluate an overall design strategy), requirement qualification set manipulation (to determine and evaluate a local control strategy), and design object description manipulation (to determine and evaluate a local control strategy). Furthermore, depending on the level of granularity, knowledge structures within RQS modification and DOD modification may be involved to model strategic knowledge that determines and evaluates modification methods (using the specialisation of GDM described in Chapter 9).

Ohsuga states that the use of strategic knowledge is indispensable as in general the problem space of a design problem is too large to handle [Ohsuga, 1997]. He distinguishes three types of strategic knowledge: knowledge to guide the decomposition of design problems, knowledge to guide exploration within the problem space, and knowledge to define the scope of the search within the problem space. By using strategic knowledge to define an effective solution space that is as small as possible, the amount of unavoidable user interaction is minimised. Ohsuga distinguishes a second role of strategic knowledge, which is to achieve a better quality of the solution: by proposing a better decomposition of the design problem, a better solution may be reached.

In GDM, strategic knowledge to guide the decomposition of design problems and the exploration within the problem space as well as to define the scope of the search within the problem space is modelled by knowledge structures within RQS modification and, depending on the level of granularity, more specifically within RQS modification determination.

Examples of design strategies. This chapter gave examples of design strategies for a design process as a whole (e.g., "Follow a practical approach"), for a requirement qualification set manipulation process (e.g., "Ignore the client's initial soft requirements") and for a design object description manipulation process (e.g., "Use an existing design"). These examples were taken from the domain of house building in the Netherlands. In AI in Design research, a few examples of domain-independent design strategies can be found.

Mostow argues that, in order to understand how the design process can be controlled, it is necessary to uncover the reasoning behind a designer's decisions about what to do next and to represent this reasoning explicitly [Mostow, 1985]. As an example, he describes how to handle interacting goals.

The following design strategies are useful to handle pairs of interacting goals in a general manner.

- *Achieve-goals-sequentially*: first solve one goal, and then transform its solution to achieve the other.
- *Defer-commitments*: order the goals so as to start with whichever decisions impose fewest restrictions on the form of the solution.
- *Make-critical-decisions-first*: order the goals so as to start with whichever decisions are most constrained by the problem.
- *Merge-goals*: conjoin the two goals into a single specification and implement it.
- *Use-goal-as-selection-criterion*: use one goal as a criterion for selecting among different solutions to the other.
- *Combine-orderings*: if both goals can be used as selection criteria, combine the orderings they produce into one total or partial ordering.
- *Use-goal-to-budget*: decompose one goal into sub-goals parallel with the decomposition of the other.

The following design strategies are useful to handle co-operative goals (i.e., achieving one goal makes it easier to achieve the other).

- *Achieve-prerequisite-first*: first achieve the goal that satisfies a precondition for achieving the other.
- *Achieve-more-general-goal-first*: first achieve the goal subsuming the other.
- *Learn-by-solving-easier-goal-first*: if two goals are similar but one is harder, solve the easier goal first and learn from this experience to solve the harder goal.

The following design strategies are useful to handle competitive goals (i.e., one goal can be achieved only at the expense of the other).

- *Sacrifice-less-important-goal*: if one goal dominates the other, ignore the less important one.
- *Relax-goal*: if both goals are equally important, relax one of them to a weaker version that is compatible with the other goal.
- *Treat-as-trade-off*: if the goals are relative preferences rather than absolute predicates, treat the competitive relationship between them as a trade-off and choose a compromise solution.

The design strategies described by Mostow are all examples of local DOD manipulation control strategies, except *sacrifice-less-important-goal*, *relax-goal*, and *treat-as-trade-off*, which are local RQS manipulation control strategies. In GDM, differences in importance of goals are modelled as different qualifications of requirements.

Chapter 13

Design Rationale

To be useful during design, design support systems need to be developed on the basis of an understanding of human design processes. For the purpose of explanation and re-use of design decisions, it is particularly important for a design support system to make use of design history and design rationale. The generic design model GDM clearly specifies the role of design history and design rationale within a design process. The model provides a structure to distinguish different types of design rationale, according to the functional role these types of design rationale play in a design process. To demonstrate that GDM can be used well to model design processes that generate and (re-)use various types of design rationale, it has been used to model part of an example aircraft design process.

Publications. *This chapter is based on earlier research on the role of design rationale in design processes [Brazier, Langen and Treur, 1997].*

To be useful during design, design support systems need to be developed on the basis of an understanding of human design processes. Human designers often remember which design requirements they have previously considered, which (partial) designs were explored, and in which situation. They also often remember the reasons for rejecting and accepting modifications to (partial) sets of design requirements and (partial) designs. In other words, human designers are able to generate design rationales (i.e., their argumentation for making any kind of decision within design processes) and recollect them from history.

Therefore, design support systems must be designed to support design history and design rationale. The use of design history and rationale is particularly important for the purpose of:

- *explanation*, to justify why a particular decision has been made in a specific situation,
- *re-use*, to retrieve design decisions that were proficient in earlier, similar situations and take an analogous decision in the current situation.

To accommodate the use of design rationale, the generic design model GDM clearly specifies the role of design history and design rationale within a design process. For example, for both a requirement qualification set manipulation process and a design object description manipulation process, GDM distinguishes a process for maintaining a record of the manipulation process, which can be consulted to retrieve various types of information, including design rationale generated in earlier states of the design process (or even in earlier design processes). The model provides a structure to distinguish different types of design rationale, according to the functional role they play in a design process: design process co-ordination, requirement qualification set manipulation or design object description manipulation. In addition, the specialisation of GDM described in Chapter 9 shows more fine-grained design rationale on the level of the focus of modification (of the current requirement qualification set or current design object description) and the method of modification to be applied.

This chapter shows how GDM and its specialisations described in Chapter 9 have been used to model part of an example design process. The example concerns the real-life design of a Fokker aircraft, the Fokker 60. The example demonstrates that GDM can be used well to model design processes that generate and (re-)use various types of design rationale.

This chapter is organised as follows. Section 13.1 discusses related work on design rationale. In Section 13.2, GDM is used to analyse the reasoning steps made in the example aircraft design process. Section 13.3 analyses the generation, reuse and storage of design rationale during the example aircraft design process, resulting in a classification of different types of design rationale on the basis of GDM. Finally, Section 13.4 discusses our approach.

13.1 Related Work

Research into design rationale is partly based on research of how human designers design. Models of human design behaviour are often the result of analyses of design tasks, design processes and designers' approaches. These models are based on various disciplines such as Cognitive Science, Mechanical Engineering, Artificial Intelligence and Artificial Intelligence in Design (e.g., [Akin, 1978; Schön, 1983; Pahl and Beitz, 1984; Brown and Chandrasekaran, 1989; Chandrasekaran, 1990; Smithers, Corne and Ross, 1994]).

Prevalent subjects of research are the *representation* of design rationale (e.g., [Chung and Goodwin, 1994; Ganeshan, Garret and Finger, 1994]), the *capture* of design rationale (e.g., [Gruber, Baudin, Boose and Weber, 1991; Klein, 1992; Candy and Edmonds, 1994; Candy, Edmonds and Patrick, 1995]) and the *use* of design rationale (e.g., [Mostow, 1989; McKerlie and McLean, 1994; Burge and Brown, 2000]). There is also research which attempts to integrate these three subjects into one framework (e.g., [Gruber and Russel, 1990]).

Besides research into design rationale, there is also related research that has resulted in services that can be used to represent and use design rationale. For example, Petrie, Cutkosky and Park have developed the *Redux'* server that offers design co-ordination services (such as context maintenance) to distributed design agents [Petrie, Cutkosky and Park, 1994]; this server can be used to represent and maintain design rationale in multi-agent environments.

The research reported in this chapter mainly focuses on the use of design rationale within a design process and (since GDM is a knowledge-level model) only for a small part on the representation of design rationale.

13.2 An Example of Aircraft Re-Design Using Design Rationale

In this chapter, a small part of a real-life design process is described, which illustrates the way in which the generic design model GDM can be used to model, specify and re-use design rationale. The example concentrates on the design of emergency exits of a new aircraft for 60 passengers, the Fokker 60 (Fo60). In order to lower product development costs, production costs and time-to-market, the design of the Fo60 was based on that of its successful predecessor, the Fo50Inc. Especially air-worthiness of a new type of aircraft is much faster to establish if its design is based on that of a type of aircraft that has already proven to be air-worthy. For the description of the example design process of the Fo60, the rationale of the design of the Fo50Inc was available.

13.1.1 Aircraft design requirements

During aircraft design, different types of requirements are imposed: requirements that have to be satisfied by all aircraft (e.g., safety regulations), requirements of the specific type of aircraft (e.g., the number of seats), and meta-requirements imposed on the design process (e.g., the deadline for finishing the design). These requirements may be difficult to satisfy simultaneously: spending more time on safety measures, for instance, may cause the design project to last beyond the deadline. Therefore, rather than designing from scratch it makes sense to use an existing design as a point of departure. By assuming that the existing design satisfies all design requirements of the aircraft to be constructed (general as well as aircraft specific), initially only those design requirements of the existing design need to be reconsidered that are in conflict with the new requirements. Furthermore, only attention has to be paid to those parts of the existing design that violate the new requirements.

General requirements such as limited product development time and limited production cost determine the overall strategy for designing the aircraft. In addition, the number of aircraft for which the design is expected to break even (with regards to return of investment) influences the overall strategy employed. For the Fo60 aircraft, the overall strategy is to keep extra costs (i.e., product development costs and production costs) to a minimum. This implies that the original design of the Fo50Inc is to be followed as closely as possible.

Table 13.1 lists the observed order in which design requirements of the Fo60 aircraft were introduced into the current requirement qualification set (marked '+') and withdrawn from the current requirement qualification set (marked '–'). The names of the qualified requirements all start with 'QR' followed by digits and/or letters. The digits are used to indicate conjunctive (simultaneous) refinements of design requirements (e.g., QR2131 and QR2132 are conjunctive refinements of QR213: both of them need to be satisfied in order to satisfy QR213). The letters are used to denote disjunctive (alternative) refinements (e.g., QR114a and QR114b are disjunctive refinements of QR114: at least one of them needs to be satisfied in order to satisfy QR114). Dotted lines separate the different steps within the trace of modifications to the set of design requirements of the Fo60.

TABLE 13.1. Part of the trace of modifications to the set of design requirements of the Fo60 aircraft.

Qualified req.	*Meaning*	*Description of requirements (when introduced)*
+QR0	R0 must be satisfied.	R0: The aircraft seats 60 passengers.
+QR1	R1 must be satisfied.	R1: The aircraft resembles the Fo50Inc aircraft as closely as possible.
+QR2	R2 must be satisfied.	R2: The return-of-investment (ROI) break-even point is at N aircraft (where N is a fixed number).
+QR3	R3 must be satisfied.	R3: All aircraft safety regulations are respected.
+QR31	R31 must be satisfied.	R31: Each side of the aircraft has at least one emergency exit.
+QR32	R32 must be satisfied.	R32: Each emergency exit is large enough to allow passengers to leave the aircraft within 90 seconds.
+QR321	R321 must be satisfied.	R321: Each emergency exit has an accessible opening of at least 20" by 36".
+QR6	R6 must be satisfied.	R6: Passengers feel comfortable within the aircraft.
+QR61	R61 must be satisfied.	R61: Safety precautions for emergency landings are not directly visible for passengers.
+QR7	R7 must be satisfied.	R7: The emergency exits are easily accessible.
+QR8	R8 must be satisfied.	R8: In emergency situations, passengers do not have to follow complicated procedures.
+QR11	R11 must be satisfied.	R11: All aircraft dimensions are the same as for the Fo50Inc.
+QR111	R111 must be satisfied.	R111: The emergency hatch functions as an emergency exit.
+QR112	R112 must be satisfied.	R112: The step-up height of the emergency hatch is 18".
+QR113	R113 must be satisfied.	R113: The size of the emergency hatch is 20" by 37.5".
+QR114	R114 must be satisfied.	R114: The emergency exits are all above the waterline (i.e., the fictitious horizontal line on the aircraft's hull below which the aircraft is lying in water after a correctly performed emergency landing on water).
+QR114a	R114a must be satisfied.	R114a: The height of the waterline is at most L_1 (where L_1 is a fixed number).

Qualified req.	*Meaning*	*Description of requirements (when introduced)*
+QR115	R115 must be satisfied.	R115: The aircraft seats 48 passengers.
+QR21	R21 must be satisfied.	R21: Only minimal changes are applied to the design.
+QR211	R211 must be satisfied.	R211: Only minimal changes are applied to the structure.
+QR212	R212 must be satisfied.	R212: Only minimal changes are applied to the shape.
+QR213	R213 must be satisfied.	R213: Only minimal changes are applied to the production method.
+QR2131	R2131 must be satisfied.	R2131: Only minimal changes are applied to the tooling.
+QR2132	R2132 must be satisfied.	R2132: Only minimal changes are applied to the material.
+QR4	R4 must be satisfied.	R4: When the aircraft is lying in water and one emergency exit is open, the aircraft does not sink within M minutes (where M is a fixed number).
+QR41	R41 must be satisfied.	R41: When the aircraft is lying in water, the opening of each emergency exit remains above water.
+QR411	R411a is preferred over R411b.	R411a: Each emergency exit is entirely above the waterline. R411b: The top edge of each emergency exit's water barrier is above the waterline.
+QR411a	R411a must be satisfied.	-
+QR5	R5 must be satisfied.	R5: Ergonomic criteria are respected.
+QR51	R51 must be satisfied.	R51: The step-up height of the emergency exit is at most 20".
–QR115	R115 must be satisfied.	R115: The aircraft seats 48 passengers.
–QR114a	R114a must be satisfied.	-
+QR114b	R114b must be satisfied.	R114b: The height of the waterline is at most L_2 (where L_2 is a fixed number).
–QR411a	R411a must be satisfied.	-
+QR411b	R411ba is preferred over R411bb.	R411ba: The water barrier of each emergency exit is placed purely inside the aircraft, and the top edge of the water barrier is above the waterline. R411bb: The water barrier of each emergency exit is not placed purely inside the aircraft, and the top edge of the water barrier is above the waterline.
+QR411ba	R411ba must be satisfied.	-
–QR411ba	R411ba must be satisfied.	-
+QR411bb	R411bba is preferred over R411bbb.	R411bba: The water barrier of each emergency exit is placed purely outside the aircraft, and the top edge of the water barrier is above the waterline. R411bbb: The water barrier of each emergency exit is placed not purely inside or purely outside the aircraft, and the top edge of the water barrier is above the waterline.
+QR411bba	R411bba must be satisfied.	-

The Fo50Inc has been designed on the basis of the design of the Fokker 50 (Fo50). One of the elements in the design rationale underlying the design of the Fo50Inc is:

> "The size of the emergency hatch is chosen to be 20" × 37.5" to achieve a maximum opening within requirements imposed on cost and structure for the Fo50."

This design rationale explains the existence of qualified requirement QR113.

General requirements have been refined either on the basis of knowledge of requirements imposed during the design of the Fo50Inc or new requirements imposed by the designer of the Fo60. For example, one way to achieve safety requirement QR41, stating that the opening of each emergency exit must remain above water, is to position the emergency exit entirely above the waterline (R411a); this requirement was imposed during the design of the Fo50Inc.

13.2.2 The example aircraft design process

On the basis of the design requirements of the Fo60 aircraft presented in Table 13.1 (which only represents a very small number of the total number of design requirements), this section describes the example aircraft design process. To demonstrate how GDM has been used to analyse this specific process, steps of this process are accompanied by references to components of GDM and its specialisations described in Chapter 9. Even for this example process, the full trace of steps is quite long; therefore, only the interesting parts are shown. Furthermore, only the qualified requirements are mentioned, neglecting the associated requirements.

After the start of the design process, first the given design requirements of the Fo60 are analysed. (In Table 13.1, these design requirements, QR0 to QR8, are listed above the first dotted line.) The design requirement QR1 states that the aircraft should resemble the Fo50Inc aircraft as closely as possible, which is analysed to be too imprecise as a basis for devising a satisfactory design object description. As a refinement of QR1, the design requirement QR11 is proposed, which states that the aircraft dimensions of the Fo60 should be the same as those of the Fo50Inc. Although this design requirement is more precise than QR1, it is still too imprecise. As a refinement of QR11, (a sub-set of) the design requirements of the Fo50Inc are proposed. (In Table 13.1, these design requirements, QR111 to QR51, are listed between the second and third dotted line.)

The most obvious contradiction in the resulting set of design requirements is that the Fo50Inc was designed to seat 48 passengers (QR115), and the new aircraft Fo60 has to seat 60 passengers (QR0). This conflict in requirements is resolved by withdrawing the Fo50Inc design requirement to seat 48. After resolving this conflict, there are no more apparent conflicts between the design requirements.

Table 13.2 shows how these steps are modelled by means of GDM, by listing the results of the activation of sub-components of its component RQSM. Note that also the specialisation of requirement qualification set manipulation as described in Chapter 9 has been used.

TABLE 13.2. First activation of the component RQSM.

Step	*Results*	*Sub-component*
r1.1.1	QR1 has to be refined.	RQS-modification-analysis
r1.1.2	The current modification focus is {QR1}.	RQS-modification-focus-determination
r1.1.3	The current modification method is extension-by-modification.	RQS-modification-method-determination
r1.1.4	QR11 is selected to be added to the current set.	RQS-modification-method-execution
r1.1.5	QR11 is added to the current set.	current-RQS-maintenance
r1.1.6	The current set and the design rationale are stored.	RQSM-history-maintenance
r1.2.1	QR11 has to be refined.	RQS-modification-analysis
r1.2.2	The current modification focus is {QR11}.	RQS-modification-focus-determination
r1.2.3	The current modification method is extension-by-retrieval.	RQS-modification-method-determination
r1.2.4	The design requirements of the Fo50Inc are selected to be retrieved and added to the current set.	RQS-modification-method-execution
r1.2.5	The design requirements of the Fo50Inc are retrieved.	RQSM-history-maintenance
r1.2.6	The design requirements of the Fo50Inc are added to the current set.	current-RQS-maintenance
r1.2.7	The current set and the design rationale are stored.	RQSM-history-maintenance
r1.3.1	QR0 and QR115 are in conflict, since QR0 states that the aircraft should seat 60 passengers and QR115 states that the aircraft should seat 48 passengers.	RQS-modification-analysis
r1.3.2	The current modification focus is {QR0, QR115}.	RQS-modification-focus-determination
r1.3.3	The current modification method is reduction-by-modification.	RQS-modification-method-determination
r1.3.4	The design requirement of the Fo50Inc, QR115, is selected to be deleted from the current set.	RQS-modification-method-execution
r1.3.5	QR115 is deleted from the current set.	current-RQS-maintenance
r1.3.6	The current set and the design rationale are stored.	RQSM-history-maintenance
r1.4.1	The design requirements in the current set are sufficiently refined and there are no apparent conflicts.	RQS-modification-analysis
r.1.4.2	The design rationale for termination is stored.	RQSM-history-maintenance

For the design of the Fo60, the design object description manipulation process starts by retrieving the design of the Fo50Inc. A problem that is subsequently encountered is that in case of an emergency landing on water, the 60-seat aircraft will lie deeper in water than the Fo50Inc, due to increase in weight. This not only leads to a conflict with QR114a (specifying a maximum waterline height that held for the Fo50Inc), but also with QR411a (specifying that the emergency exits must be entirely above the waterline).

Table 13.2 shows how these steps are modelled by means of GDM, by listing the results of the activation of sub-components of its component DODM. Note that also the specialisation of design object description manipulation as described in Chapter 9 has been used.

TABLE 13.3. First activation of the component DODM.

Step	*Results*	*Sub-component*
d1.1.1	The current design is empty and neither satisfies nor violates any of the current design requirements.	DOD-modification-analysis
d1.1.2	The current modification focus is the whole design.	DOD-modification-focus-determination
d1.1.3	The current modification method is extension-by-retrieval followed by deductive-refinement*.	DOD-modification-method-determination
d1.1.4	The Fo50Inc design is selected to extend the current design. Deduction targets are generated about various properties of the aircraft, such as maximum take-off weight, waterline height and production costs.	DOD-modification-method-execution
d1.1.5	The contents of the Fo50Inc design are added to the current design.	current-DOD-maintenance
d1.1.6	From the current design, various properties of the aircraft are derived in accordance with the given deduction targets.	deductive-DOD-refinement
d1.1.7	The new deduced information is added to the current design.	current-DOD-maintenance
d1.1.8	The current design and the design rationale are stored.	DODM-history-maintenance
d1.2.1	The current design satisfies all design requirements except QR0.	DOD-modification-analysis
d1.2.2	The current modification focus is the layout of seats.	DOD-modification-focus-determination
d1.2.3	The current modification method is reduction-by-modification followed by extension-by-modification and deductive-refinement*.	DOD-modification-method-determination
d1.2.4	A new layout with 60 seats is generated as a substitute for the 48-seat layout. Deduction targets are generated for various aircraft properties.	DOD-modification-method-execution
d1.2.5	The current design is updated by deleting the 48-seat layout and adding the 60-seat layout.	current-DOD-maintenance
d1.2.6	From the current design, the new maximum take-off weight, waterline height (1.5" higher) and production costs are derived.	deductive-DOD-refinement
d1.2.7	The new deduced information is added to the current design.	current-DOD-maintenance
d1.2.8	The current design and the design rationale are stored.	DODM-history-maintenance

Step	*Results*	*Sub-component*
d1.3.1	The current design violates QR114a and QR411a. The design requirements QR0, QR111, QR112, QR113, QR114a and QR411 are found to be inconsistent, so there is no possibility to make a satisfactory design.	DOD-modification-analysis
d.1.3.2	The design rationale for termination is stored.	DODM-history-maintenance

*) Note that execution of this modification method requires more than one step within the manipulation process. For the sake of brevity, these steps have been summarised within this trace.

To remedy the violation of the design requirement QR114a (specifying a maximum waterline height that held for the Fo50Inc), requirement qualification set manipulation replaces QR114a by the new design requirement QR114b, which specifies a maximum waterline height that is equal to the waterline height in the current design. Table 13.4 shows how these steps are modelled.

TABLE 13.4. Second activation of the component RQSM.

Step	*Results*	*Sub-component*
r2.1.1	QR114a cannot be satisfied without violating QR0, QR111, QR112, QR113 or QR411.	RQS-modification-analysis
r2.1.2	The current modification focus is {QR114a, QR0, QR111, QR112, QR113, QR411}.	RQS-modification-focus-determination
r2.1.3	The current modification method is reduction-by-modification followed by extension-by-modification.	RQS-modification-method-determination
r2.1.4	As a substitute of QR114a, a new design requirement QR114b is generated, expressing a maximum height of the waterline that is equal to the waterline height in the current design.	RQS-modification-method-execution
r2.1.5	The current set is updated by deleting QR114a and adding QR114b.	current-RQS-maintenance
r2.1.6	The current set and the design rationale are stored.	RQSM-history-maintenance
r2.2.1	The design requirements in the current set are sufficiently refined and there are no apparent conflicts.	RQS-modification-analysis
r.2.2.2	The design rationale for termination is stored.	RQSM-history-maintenance

To satisfy QR411a, design object description manipulation devises a new partial design in which the emergency exit is placed entirely above the new waterline. However, the resulting design is in conflict with the design requirements QR211, QR2131 and QR2132 (which are all refinements of QR21, specifying that only minimal changes should be applied to the design). Table 13.5 shows how these steps are modelled.

TABLE 13.5. Second activation of the component DODM.

Step	*Results*	*Sub-component*
d2.1.1	All design requirements are satisfied except QR411a, which is violated.	DOD-modification-analysis
d2.1.2	The current modification focus is the sub-design of the emergency exits.	DOD-modification-focus-determination
d2.1.3	The current modification method is reduction-by-modification followed by extension-by-modification and deductive-refinement.	DOD-modification-method-determination
d2.1.4	A new position for the emergency exits is generated that is 1.5" higher than in the current design. Deduction targets are generated about various aircraft properties.	DOD-modification-method-execution
d2.1.5	The current design is updated by substituting the sub-design of the emergency exits by the sub-design in which the emergency exists are positioned 1.5" higher.	current-DOD-maintenance
d2.1.6	From the current design, the new production costs for the aircraft are derived (which are different from the previous design, because of the changed position of the emergency exits).	deductive-DOD-refinement
d2.1.7	The new deduced information is added to the current design.	current-DOD-maintenance
d2.1.8	The current design and the design rationale are stored.	DODM-history-maintenance
d2.2.1	The cost-related design requirements QR211, QR213 and QR2131 are violated.	DOD-modification-analysis
d2.2.2	The current modification focus is the whole design.	DOD-modification-focus-determination
d2.2.3	The current modification method is reduction-by-modification followed by extension-by-retrieval and deductive-refinement	DOD-modification-method-determination
d2.2.4	The contents of the current design are selected to be deleted. The design stored in step d1.1.8 is selected for retrieval, as this is the most recently stored design that does not violate design requirement QR411a. Deduction targets are generated about various aircraft properties.	DOD-modification-method-execution
d2.2.5	The design stored in step d1.1.8 is retrieved.	DODM-history-maintenance
d2.2.6	The contents of the current design are replaced by the contents of the retrieved design.	current-DOD-maintenance
d2.2.7	From the current design, various properties of the aircraft are derived.	deductive-DOD-refinement

Step	*Results*	*Sub-component*
d2.2.8	The new deduced information is added to the current design.	current-DOD-maintenance
d2.2.9	The current design and the design rationale are stored.	DODM-history-maintenance
d2.3.1	Again, the cost-related design requirements QR211, QR213 and QR2131 are violated. No earlier design can be found satisfying design requirement QR411a, so there is no possibility to make a satisfactory design.	DOD-modification-analysis
d.2.3.2	The design rationale for termination is stored.	DODM-history-maintenance

The design requirement that the emergency exit should remain entirely above the waterline (QR411a) cannot be satisfied. Therefore, this design requirement is withdrawn and a new design requirement is introduced, namely that the top edge of each emergency exit's water barrier should be above the waterline (QR411b). Since QR411b does not state where the water barrier has to be placed, it has to be refined. There are several types of water barriers; on the basis of knowledge of expected cost of a solution, the first option considered for a design requirement on the water barrier is to have a water barrier on the inside of the aircraft (QR411ba). Table 13.6 shows how these steps are modelled.

TABLE 13.6. Third activation of the component RQSM.

Step	*Results*	*Sub-component*
r3.1.1	It is not possible to satisfy both QR411a and the cost requirements in the current set.	RQS-modification-analysis
r3.1.2	The current modification focus is {QR411a, QR211, QR213, QR2131}.	RQS-modification-focus-determination
r3.1.3	The current modification method is reduction-by-modification followed by extension-by-modification.	RQS-modification-method-determination
r3.1.4	QR411a is selected to be deleted and a new design requirement QR411b is generated to be added.	RQS-modification-method-execution
r3.1.5	The current set is updated by deleting QR411a and adding QR411b.	current-RQS-maintenance
r3.1.6	The current set and the design rationale are stored.	RQSM-history-maintenance
r3.2.1	QR411b has to be refined.	RQS-modification-analysis
r3.2.2	The current modification focus is {QR411b}.	RQS-modification-focus-determination
r3.2.3	The current modification method is extension-by-modification.	RQS-modification-method-determination

Step	*Results*	*Sub-component*
r3.2.4	Design requirement QR411b can be refined in two ways: either by QR411ba (a water barrier inside the aircraft) or by QR411bb (a water barrier not inside the aircraft). Because the first option is expected to be cheaper, this refinement is selected.	RQS-modification-method-execution
r3.2.5	QR411ba is added to the current set.	current-RQS-maintenance
r3.2.6	The current set and the design rationale are stored.	RQSM-history-maintenance
r3.3.1	There are no further apparent conflicts.	RQS-modification-analysis
r3.3.2	The design rationale for termination is stored.	RQSM-history-maintenance

Based on knowledge of the solution to the problem of designing a water barrier for the passenger door of the Fo50 (which is inherited by the Fo50Inc), a new sub-design is devised with a separate, 1.5” high panel to be placed in the opening of the emergency exit. This solution, however, is in conflict with the requirement that safety precautions for emergency landings should not be visible (QR61) and that passengers cannot be expected to follow complicated procedures in emergency situations (QR8). Therefore, the most recent design is restored that does not violate design requirement QR411ba. Subsequently, it is concluded that there is no way to generate a design that satisfies QR411ba, QR61 and QR8 simultaneously. Table 13.7 shows how these steps are modelled.

TABLE 13.7. Third activation of the component DODM.

Step	*Results*	*Sub-component*
d3.1.1	All design requirements are satisfied by the current design, except QR411b and QR411ba (which are neither satisfied nor violated).	DOD-modification-analysis
d3.1.2	The current modification focus is the sub-design of the emergency exits.	DOD-modification-focus-determination
d3.1.3	The modification method chosen is extension-by-modification followed by deductive-refinement.	DOD-modification-method-determination
d3.1.4	As a solution for QR411ba, a modification is generated involving a panel of 1.5" high to be placed in the emergency exit. Deduction targets are generated about various aircraft properties.	DOD-modification-method-execution
d3.1.5	The sub-design of a 1.5" high panel that can be placed in the emergency exit is added to the current design.	current-DOD-maintenance
d3.1.6	It is deduced that a panel to be placed in the emergency exit is visible for passengers and that it requires complicated procedures for passengers to place it.	deductive-DOD-refinement

Step	*Results*	*Sub-component*
d3.1.7	The new deduced information is added to the current design.	current-DOD-maintenance
d3.1.8	The current design and the design rationale are stored.	DODM-history-maintenance
d3.2.1	The design requirements QR8, QR6 and QR61 are violated.	DOD-modification-analysis
d3.2.2	The current modification focus is the whole design.	DOD-modification-focus-determination
d3.2.3	The current modification method is reduction-by-modification followed by extension-by-retrieval and deductive-refinement.	DOD-modification-method-determination
d3.2.4	The contents of the current design are selected to be deleted. The design stored in step d2.1.8 is selected for retrieval, as this is the most recently stored design that does not violate design requirement QR411b. Deduction targets are generated about various aircraft properties.	DOD-modification-method-execution
d3.2.5	The design stored in step d2.1.8 is retrieved.	DODM-history-maintenance
d3.2.6	The contents of the current design are replaced by the contents of the retrieved design.	current-DOD-maintenance
d3.2.7	From the current design, various properties of the aircraft are derived.	deductive-DOD-refinement
d3.2.8	The new deduced information is added to the current design.	current-DOD-maintenance
d3.2.9	The current design and the design rationale are stored.	DODM-history-maintenance
d3.3.1	The design requirement QR411ba and the design requirements on passenger comfort and safety are found to be inconsistent, so there is no possibility to make a satisfactory design.	DOD-modification-analysis
d3.3.2	The design rationale for termination is stored.	DODM-history-maintenance

The designer decides to reconsider the design requirement that the water barrier should be placed on the inside of the aircraft. The designer is aware of the option of designing a water barrier in-between the interior panels and the skin of the aircraft, but decides to consider water barriers not inside the aircraft first (QR411bb), in view of the expected cost implications. Since QR411bb does not state where the water barriers have to be placed (except that they should not be inside the aircraft), it has to be refined. There are several types of water barriers that can be chosen; the first option considered for a design requirement is to have a water barrier on the outside of the aircraft (QR411bba). Table 13.8 shows how these steps are modelled.

TABLE 13.8. Fourth activation of the component RQSM.

Step	*Results*	*Sub-component*
r4.1.1	It is not possible to satisfy both QR411ba and the design requirements on passenger comfort and safety in the current set.	RQS-modification-analysis
r4.1.2	The current modification focus is {QR411ba, QR8, QR6, QR61}.	RQS-modification-focus-determination
r4.1.3	The current modification method is reduction-by-modification followed by extension-by-retrieval.	RQS-modification-method-determination
r4.1.4	Design requirement QR411ba was one of two options to refine QR411b (see step r3.2.4). Since this option is untenable, the other option, QR411bb, is selected.	RQS-modification-method-execution
r4.1.5	The current description is updated by deleting QR411ba and adding QR411bb.	current-RQS-maintenance
r4.1.6	The current set and the design rationale are stored.	RQSM-history-maintenance
r4.2.1	QR411bb has to be refined.	RQS-modification-analysis
r4.2.2	The current modification focus is {QR411bb}.	RQS-modification-focus-determination
r4.2.3	The current modification method is extension-by-modification.	RQS-modification-method-determination
r4.2.4	Design requirement QR411bb can be refined in two ways: either by QR411bba (a water barrier outside the aircraft) or by QR411bbb (a water barrier not purely inside or outside the aircraft). For the first option, solutions are known from experience and knowledge acquired in the past. Therefore, this option is chosen.	RQS-modification-method-execution
r4.2.5	QR411bba is added to the current set.	current-RQS-maintenance
r4.2.6	The current set and the design rationale are stored.	RQSM-history-maintenance
r4.3.1	There are no further apparent conflicts.	RQS-modification-analysis
r4.3.2	The design rationale for termination is stored.	RQSM-history-maintenance

One option for a water barrier that is outside the aircraft is an airbag, which is a solution based on the designer's knowledge of other aircraft—the designer believes that this the best solution, given the current design requirements. Subsequently, the implications of the design for the aerodynamics of the aircraft are explored, and so on, and so forth. Table 13.9 shows how these steps are modelled.

TABLE 13.9. Fourth activation of the component DODM.

Step	*Results*	*Sub-component*
d4.1.1	All design requirements are satisfied, except QR411bb and QR411bba, which are neither satisfied nor violated.	DOD-modification-analysis
d4.1.2	The current modification focus is the sub-design of the emergency exits.	DOD-modification-focus-determination
d4.1.3	The current modification method is extension-by-retrieval followed by deductive-refinement.	DOD-modification-method-determination
d4.1.4	The sub-design of an airbag outside the emergency exit is selected for retrieval. Deduction targets are generated about various aircraft properties.	DOD-modification-method-execution
d4.1.5	The sub-design of an airbag outside the emergency exit is retrieved.	DODM-history-maintenance
d4.1.6	The sub-design of an airbag outside the emergency exit is added to the current design.	current-DOD-maintenance
d4.1.7	From the current design, various properties of the aircraft such as aerodynamics are derived.	deductive-DOD-refinement
d4.1.8	The new deduced information is added to the current design.	current-DOD-maintenance
d4.1.9	The current design and the design rationale are stored.	DODM-history-maintenance
…	…	…

13.3 Rationale Used in the Aircraft Re-design Example

Design rationale is the argumentation of decisions made during a design process: the options considered, and the decision criteria for choosing a specific option. Design rationale is often re-used during a design process, since a new decision can be made on the basis of this information besides the information about the current state of the process. Analysis of the design rationale of an earlier decision in view of the current situation can show, for example, which options still have to be considered or which decision criteria no longer hold.

In the example presented in Section 13.2, design rationale is generated during the design process and stored for later use. (Note that the design rationale for decisions made within the design process co-ordination process has been left out from the example.) In this section, this design rationale is identified and classified. Section 13.3.1 discusses the design rationale used for decisions made within the requirement qualification set manipulation process. Section 13.3.2 discusses the design rationale used for decisions made within the design object description manipulation process.

13.3.1 Design rationale used for decisions within the RQS manipulation process

Table 13.10 presents the design rationale used during the requirement qualification set manipulation process in the example design process presented in Section 13.2. Table 13.10 shows the sub-component of the component RQSM in which the design rationale is used, and a number of occurrences in the example design process.

TABLE 13.10. Design rationale used within the component RQSM.

Design rationale	*Used by sub-component*	*Used in step(s)*
At the start of an activation, or each time the current set changes, determine conflicts between design requirements and insufficiently refined design requirements.	RQS-modification-analysis	r1.1.1, r1.2.1, r1.3.1, r1.4.1, r2.1.1, r2.2.1, r3.1.1, r3.2.1, r3.3.1, r4.1.1, r4.2.1, r4.3.1
If all design requirements in the current set are sufficiently refined and there are no apparent conflicts, then there is no need to modify the current set any further.	RQS-modification-analysis	r1.4.1, r2.2.1, r3.3.1, r4.3.1
If there is a conflict between design requirements, then focus on the involved design requirements.	RQS-modification-focus-determination	r1.3.2, r2.1.2, r3.1.2, r4.1.2
If there are insufficiently refined design requirements in the current set, but no (apparent) conflicts between the design requirements, then focus on the insufficiently refined design requirements.	RQS-modification-focus-determination	r1.1.2, r1.2.1, r3.2.2, r4.2.2
If the focus is on insufficiently refined design requirements, then choose to extend the current set with new design requirements.	RQS-modification-method-determination	r1.1.3, r1.2.3, r3.2.3, r4.2.3
If the focus is on conflicting design requirements, then choose to replace one of these design requirements by a new design requirement.	RQS-modification-method-determination	r2.1.3, r3.1.3, r4.1.3
If an earlier design is re-used and design requirements for that design are in conflict with new design requirements, then delete the old design requirements.	RQS-modification-method-execution	r1.3.4
If there is a choice between two or more design requirements to be replaced, then replace the design requirement with the smallest impact on the design (using a heuristic to determine the expected impact).	RQS-modification-method-execution	r2.1.4, r3.1.4
When reusing earlier designs, if old requirements are untenable with respect to the current design, then replace them by alternatives tuned to the current design.	RQS-modification-method-execution	r2.1.4

Design rationale	*Used by sub-component*	*Used in step(s)*
If more than one refinement of a given design requirement is possible, then choose the one that least jeopardises the chances to satisfy other design requirements (using a heuristic to determine the expected impact).	RQS-modification-method-execution	r3.2.4, r4.2.4

13.3.2 Design rationale used for decisions within the DOD manipulation process

Table 13.11 presents the design rationale used during the design object description manipulation process in the example design process presented in Section 13.2. Table 13.11 shows the sub-component of the component DODM in which the design rationale is used, and a number of occurrences in the example design process.

TABLE 13.11. Design rationale used within the component DODM.

Design rationale	*Used by sub-component*	*Used in step(s)*
At the start of an activation, or each time the current design changes, determine design requirements that are inconsistent or that are violated or not satisfied by the current design.	DOD-modification-analysis	d1.1.1, d1.2.1, d1.3.1, d2.1.1, d2.2.1, d2.3.1, d3.1.1, d3.2.1, d3.3.1, d4.1.1
If all known modifications of the current design in order to satisfy a specific design requirement lead to the violation of another design requirement, then no (further) modifications should be attempted.	DOD-modification-analysis	d1.3.1, d2.3.1, d3.3.1
If there are non-satisfied design requirements but no violated design requirements, focus on the part of the current design that is (implicitly) referred to by these non-satisfied design requirements.	DOD-modification-focus-determination	d1.1.2, d4.1.2
If there are violated design requirements, focus on the part of the current design that is (implicitly) referred to by these violated design requirements.	DOD-modification-focus-determination	d1.2.2, d2.1.2, d2.2.2, d3.1.2, d3.2.2
If a design requirement is violated by some part of the design, then propose to replace this part and to determine the deductive refinement of the new design.	DOD-modification-method-determination	d1.2.3, d2.1.3, d2.2.3, d3.2.3
If there are non-satisfied design requirements but no violated design requirements, then propose to extend the part of the design in focus and to deductively refine the new design. For the extension, first try to use an earlier design that satisfies most of the given design requirements.	DOD-modification-method-determination	d1.1.3, d3.1.3, d4.1.3

13.4 Discussion

During design processes, designers are known to use previous 'trains of thought': sequences of design steps, and the rationale behind the reasoning involved. They continually explore new options, backtrack, and rethink previous options. To guide the design process, human experts use knowledge of design strategies and methods, together with information about the current state of the design process, their preferences, assumptions and, often, idiosyncrasies.

The generic task model GDM has been used to model an example aircraft design process, to illustrate the various instances of design rationale that are generated and used. These instances include the design decisions themselves, the available and considered options for design decisions, the criteria upon which design decisions are based, and the arguments for design decisions (e.g., reasons why a particular design decision has been made).

GDM has proven to provide a structure for distinguishing and modelling different types of design rationale, on the basis of the functional role they play in design processes. At the highest level of abstraction, GDM distinguishes decisions with respect to the overall strategy of the design process, requirement qualification set manipulation, and design object description manipulation. At lower levels of abstraction, specialisations of GDM distinguish and model a number of more specific types of design rationale, related to processes such as modification analysis, modification focus determination, modification method determination, and modification method execution.

Chapter 14

Conflict Management in Design

Conflicts are inherent to design. The view underlying conflict management in design *is that design is a process of detecting, prioritising and eventually resolving design conflicts. The complex reasoning processes involved in conflict management in design are highly dynamic and non-monotonic: resolving one conflict may cause the generation of another conflict. This chapter analyses possible types of design conflicts and presents excerpts from specialisations of the generic design model GDM that model conflict management in design.*

Publications. *This chapter is based on a publication about conflict management in design with an earlier version of GDM [Brazier, Langen and Treur, 1995*b*].*

Conflicts are inherent to design. Contradictory design requirements, incompatible interests of designers from the design team and divergent modification proposals are inevitable aspects of many design processes. In general, a *design conflict* is an inconsistency between particular elements of design information used or produced within a design process.

The view underlying *conflict management in design* is that design is a process of detecting, prioritising and eventually resolving design conflicts. In design practice, detected design conflicts are not resolved immediately: different requirements, designers' interests and designs are often considered simultaneously and explored in parallel, and at different stages different conflicts are detected, prioritised and resolved. This holds in particular for collaborative design efforts in which two or more designers collectively design a complex artefact. The co-ordination of an individual designer's design process requires extensive knowledge of strategies determining which (type of) detected conflicts should be resolved next, and thus, which ones should be resolved at a later moment. Hence, design includes not only detection and (immediate or delayed) resolution of conflicts, but also prioritisation of conflicts.

The complex reasoning processes involved in conflict management in design are highly dynamic and non-monotonic. Design conflicts often evolve in the sense that the resolution of one conflict results in the creation of another (either deliberately or involuntarily), that, in turn, may (have to) be resolved immediately or may be ignored for a while. However, for a design process to end successfully, there may be no design conflicts left, regardless of which overall design strategy is effective.

This chapter analyses possible types of design conflicts and presents excerpts from specialisations of the generic design model GDM that model conflict management in design. Section 14.1 discusses related work on conflict management. Section 14.2 presents a typology of design conflicts based on GDM, illustrated by means of examples adopted from building design. Sections 14.3 to 14.5 present how GDM can be extended and used to model and specify the detection, prioritisation and resolution of RQS manipulation conflicts, DOD manipulation conflicts, and design process co-ordination conflicts, respectively. Finally, Section 14.6 discusses the main contributions of this chapter.

14.1 Related Work

Conflicts exist within each and every design process, whether it is an individual agent's design process or a collaborative design effort involving multiple design agents, possibly with expertise in different domains. This section describes how researchers from AI in Design have addressed the problem of providing computational support for conflict management in design support systems, both for single-agent and multi-agent design processes. The section ends with a discussion of the main contributions of their work and issues left open. The five approaches discussed in this section are:

- conflict management by design and resolution,
- conflict management by integrated exception handling,
- conflict management by tracking Pareto optimality,
- conflict management by negotiation between single-function agents,
- conflict management by assumption-based constraint satisfaction.

14.1.1 Conflict management by design and resolution

Oh and Sharpe delineate the following sources of conflict that are inherent in an interdisciplinary design environment [Oh and Sharpe, 1995]:

1. Differences in technical beliefs held by designers.
2. Differences in decisions and assumptions made by designers.
3. Differences in technical vocabulary used by designers from different domains.
4. Lack of common and shared understanding between designers.

5. Inconsistencies between design models used by designers.
6. Differences in goals or preferences maintained by designers.
7. Differences between evaluations of criteria used by designers.
8. Differences between solution components suggested by designers.

Their Schemebuilder environment provides support for the creation of technologies and enables the comparison of technological alternatives before major commitments are made and irrevocable decisions are taken. Schemebuilder uses two basic approaches to managing design conflicts: (1) avoid them or minimise their occurrence (which Oh and Sharpe call conflict management by design) and (2) resolve them at run-time when they occur.

The first approach addresses, to some extent, conflict situations of differences in technical beliefs, differences in technical vocabulary, lack of common and shared understanding, and inconsistencies between design models. To this end, Schemebuilder provides a knowledge browsing system for accessing and sharing technological information, bond graphs for a common vocabulary, and a group data management system to limit inconsistencies between design models.

To support the second approach, Schemebuilder provides multi-criteria decision-making techniques, some failure-handling mechanisms with rule-oriented strategies and conventions, strategies for constraint relaxation and adjustment, and a negotiation protocol.

14.1.2 Conflict management by integrated exception handling

Klein views conflict management within a collaborative design environment as the exception handling component of an integrated set of collaborative design co-ordination services [Klein, 1995]. He describes the following list of conflicts that a design co-ordination system must be able to handle:

1. Changes in task requirements, organisational policies, or resources.
2. Errors, such as incorrect results, late results, decision conflicts, inter-process resource conflicts and communication breakdowns.
3. Missed opportunities (i.e., failures to capitalise on an unexpected opportunity).

For this purpose, Klein proposes a design co-ordination system, iDCCS, that provides three services: (1) a dependency capture service to capture all process, product, and organisational decisions and their interdependencies, (2) a process enactment service to enable messaging, project management, structured conversation and workflow, and (3) an exception management service to anticipate and detect exceptions that occur during process enactment, and to suggest strategies for resolving these exceptions, making extensive use of the information collected by the dependency capture service. Exception handling strategies work by modifying process and product decisions, including assignment decisions, task sequencing, design components, and so on.

14.1.3 Conflict management by tracking Pareto optimality

Petrie, Webster and Cutkosky do not distinguish different types of design conflict, but they focus on conflicts that can be mapped onto objectives and constraints in an integer programming (IP) approach [Petrie, Webster and Cutkosky, 1995]. To satisfy multiple objectives among distributed agents, Petrie *et al.* use a formalism of tracking the Pareto optimality of a design solution. (They define a Pareto optimal design solution as a solution that cannot be improved with respect to a global objective function without making some other part of the solution worse.) Tracking in this sense means that the problem solver is automatically notified of a loss of Pareto optimality and of opportunities to improve the design.

This "bookkeeping" functionality can be applied to support configuration design tasks. It has three advantages: (1) opportunities to improve the local solution are noticed that otherwise would be lost, (2) resolution of conflicts may be delayed arbitrarily, and (3) revisiting a problem-solving state known to be bad during problem solving backtracking is prevented. Petrie *et al.* have used a sub-set of the design problem solving model Redux ([Petrie, 1992]) to define *Redux'* for dependency-directed backtracking by means of Pareto optimality, and implemented it as a co-ordination service agent. The *Redux'* server does not provide conflict management strategies, but just the basic dependency tracking information that a design problem solver needs to manage conflicts within a configuration design process.

14.1.4 Conflict management by negotiation between single-function agents

For concurrent engineering tasks, Dunskus, Grecu, Brown and Berker distinguish so-called single-function agents for the functions of advice, analysis, criticism, estimation, evaluation, planning, selection and suggestion within a design process [Dunskus, Grecu, Brown and Berker, 1995]. According to this paradigm, a design process is a co-operative effort of interacting, dedicated agents.

The makeup of each individual single-function agent is meant to be simple. The following types of conflict may occur in the interactions between single-function agents:

1. Insufficient information: the information provided by one agent is insufficient for another agent that needs the information.
2. Information of insufficient quality: the quality of the information provided by one agent is insufficient for another agent that needs the information.
3. Poor processing model: the processing model used by one agent to produce information needed by another agent is too poor for that other agent.
4. Differing preferences: two agents of the same type have different preferences and therefore make different choices (suggestions or decisions) for the same problem.
5. Design constraint violation: a choice made by one agent (especially an Advisor, a Selector or a Suggestor) is turned down by another agent (especially a Critic), because of a design constraint violation caused by that choice.

To resolve such conflicts, the conflicting agents have to negotiate about the addition of needed information, the improvement of information quality, a change to a better processing model, or an agreement to undo a design constraint violation. In line with their view of conflict management as a negotiation between single-function agents, Dunskus *et al.* propose the SINE platform. By means of SINE, a reusable single-function agent system can be built, conflict-resolution knowledge can be implemented, and a negotiation language for single-function agents can be designed and implemented.

14.1.5 Conflict management by assumption-based constraint satisfaction

For preliminary engineering tasks, Haroud, Boulanger, Gelle and Smith propose to manage conflicts by means of strategies that integrate appropriate representation of assumptions with techniques for solving the conflicts which these assumptions may cause [Haroud, Boulanger, Gelle and Smith, 1995]. Haroud *et al.* distinguish two types: default assumptions (reflecting a rough statistical understanding of previous experience), and preference assumptions (reflecting the wishes to proceed in a certain way according to ill-formalised criteria).

Following the dynamic constraint satisfaction paradigm, Haroud *et al.* model design relationships as constraints and particular characteristics or dimensions of the design solution as variables. Each stage of a design process is represented by a context, consisting of a set of derived variables with their associated range of values, a set of derived constraints, and justifications of constraints and variable values (in terms of the decisions that introduced them).

Haroud *et al.* use a model of dynamic constraint satisfaction, which consists of a combination of a search process and constraint propagation. The search process derives relevant variables and constraints sequentially, guided toward alternative constraint combinations by means of default assumptions and preference assumptions. At each search level, constraint propagation propagates new constraints within the different solution regions.

Haroud *et al.* claim that the use of assumptions, translated into default and preference constraints, together with the use of fixed constraints (representing physical principles of the domain), is not only useful to guide search: it also provides a means to support conflict management strategies close to those applied by designers in practice. They describe a conflict management algorithm, in which assumption conflicts are solved either by dropping default constraints or by weakening preference constraints, while conflicts between fixed constraints result in dependency-directed backtracking. The constraint which is to be dropped or weakened, or on which to backtrack, is determined by means of constraint ranking.

14.1.6 Discussion

Three of the five approaches described above are dedicated to collaborative design, involving multiple design agents (Oh and Sharpe, Klein, and Petrie *et al.*); the approach of Haroud *et al.* assumes a single design agent. The approach of Dunskus *et al.* considers a (complex) de-

sign problem to be a multi-agent design problem, involving relatively simple single-function agents, each tailored to address only part of a design problem.

Four of the five approaches (all but Dunskus *et al.*) provide services for capturing design decisions and their interdependencies (sometimes including design rationale) for the purpose of conflict resolution. Only Oh and Sharpe pay attention to conflict prevention.

The approaches of Petrie *et al.* and Haroud *et al.* are tuned to specific types of design problems, namely those that can be modelled as integer programming problems and dynamic constraint satisfaction problems, respectively. For the other approaches, no such assumptions are made.

What is missing in these approaches is an elaborate typology of the design conflicts that a (single) design agent may encounter, as well as a thorough treatment of the complex (i.e., dynamic and non-monotonic) reasoning involved in conflict management.

14.2 A Typology of Design Conflicts

The three components of the generic task model GDM provide a basis for the distinction of three main types of conflicts encountered in an individual agent's design process: RQS manipulation conflicts, DOD manipulation conflicts, and design process co-ordination conflicts. Co-ordination conflicts between multiple design agents are the fourth main type of conflict. Table 14.1 shows a typology of these four main types of design conflicts (each shown in different columns). The remainder of this section describes each of these conflict types in detail.

14.2.1 RQS manipulation conflicts

An *RQS manipulation conflict* is a design conflict that is caused by a requirement qualification set manipulation process. There are three main types of RQS manipulation conflicts: RQS assessment conflicts, RQS modification conflicts and RQS refinement conflicts.

14.2.1.1 RQS assessment conflicts

An *RQS assessment conflict* concerns a specific requirement qualification set that cannot be fulfilled (as a whole or in part) by a consistent design object description. There are two types of RQS assessment conflicts: design requirement satisfaction conflicts, and overall RQS assessment conflicts.

A *design requirement satisfaction conflict* occurs when a specific design requirement cannot be satisfied by a consistent design object description. An *overall RQS assessment conflict* occurs when a specific requirement qualification set includes qualified requirements that cannot be satisfied by one and the same, consistent design object description.

TABLE 14.1. Possible types of conflict in a design process.

Conflict Type	*Manipulation*		*Process Co-ordination*	
	RQSM	*DODM*	*DPC*	*Agent*
1. Assessment conflict				
1.1. Satisfaction conflict	•	•		
1.1.1. Intra-view satisfaction conflict	•	•		
1.1.2. Inter-view satisfaction conflict	•	•		
1.2. Fulfilment conflict	•	•		
1.2.1. Intra-view fulfilment conflict	•	•		
1.2.2. Inter-view fulfilment conflict	•	•		
2. Modification conflict				
2.1. Modification focus conflict				
2.1.1. Modification focus generation conflict	•	•		
2.1.2. Modification focus selection conflict	•	•		
2.2. Modification method conflict				
2.2.1. Modification method generation conflict	•	•		
2.2.2. Modification method selection conflict	•	•		
2.3. Alteration conflict				
2.3.1. Alteration generation conflict	•	•		
2.3.2. Alteration selection conflict	•	•		
3. Refinement conflict				
3.1. Deductive refinement conflict	•	•		
3.2. Extension conflict	•	•		
4. Design process co-ordination conflict				
4.1. Design process objective satisfaction conflict			•	
4.2. Design strategy conflict				
4.2.1. Design strategy generation conflict			•	
4.2.2. Design strategy selection conflict			•	
5. Co-operation conflict				
5.1. Language conflict				•
5.2. Belief conflict				•
5.3. Goal conflict				•
5.4. Responsibility conflict				•
5.5. Co-operation level conflict				•

A further distinction among RQS assessment conflicts can be made on the basis of views. An *RQS view* is a sub-set of the design requirement information included in a requirement qualification set, of which the members have something in common. A commonality may be the *source* (e.g., the customer), the *nature* (e.g., fire prevention regulations) or an *aspect* of the design object to which the design requirement refers (e.g., the foundation of a building).

If a number of RQS views are defined for a given requirement qualification set, then two types of RQS assessment conflicts can be distinguished: intra-view RQS assessment conflicts and inter-view RQS assessment conflicts. An *intra-view RQS assessment conflict* occurs specifically within one RQS view, whereas an *inter-view RQS assessment conflict* occurs within a combination of two or more different RQS views. Similar definitions for *intra-view/inter-view design requirement satisfaction conflicts* as well as *intra-view/inter-view overall RQS assessment conflicts* can be given.

Example 14.1. Suppose a customer wants a bungalow with a tap in the garage, but explicitly does not want to have heating in the garage. The architect argues that there must be central heating in a garage that has a tap (because otherwise, the pipes may freeze in a cold winter). This is an overall RQS assessment conflict. If there is one RQS view for the customer's design requirements and another for the architect's design requirements, then the conflict is an inter-view overall RQS assessment conflict. ■

14.2.1.2 RQS modification conflicts

An *RQS modification conflict* is caused by an RQS modification process. There are three types of RQS modification conflicts: RQS modification focus conflicts, RQS modification method conflicts, and RQS alteration conflicts.

RQS modification focus conflicts

An *RQS modification focus* is a sub-set of the design requirement information included in a specific requirement qualification set that is to be modified (by adding new design requirements, deleting existing design requirements or changing the formulation of existing design requirements). An *RQS modification focus conflict* occurs when multiple options exist to determine a new modification focus. There are two types of RQS modification focus conflicts: RQS modification focus generation conflicts, and RQS modification focus selection conflicts.

An *RQS modification focus generation conflict* occurs when multiple proposals exist for a new RQS modification focus. An *RQS modification focus selection conflict* occurs when there is no best proposal that meets all applicable criteria to select a modification focus. (That is, either no proposal meets the criteria or there are multiple proposals that meet the criteria.)

Example 14.2. At an early stage in the process of designing a house, the customer wants to modify some of his design requirements for the dormer, even though they are not in conflict with each other. The architect, on the other hand, wants to focus on the conflicting design requirements of the living room. This is an RQS modification focus generation conflict.

To select one of these proposals, the criterion of flexibility of the design process applies, sustaining the customer's suggestion (and not the architect's), as well as the criterion of efficiency of the design process, sustaining the architect's suggestion (and not the customer's). This is an RQS modification focus selection conflict. ■

RQS modification method conflicts

An *RQS modification method* is a method for modifying the design requirement information included in a requirement qualification set. Examples of RQS modification methods are (1) the refinement of imprecise design requirements into more detailed design requirements and (2) the removal of inconsistent design requirements. An *RQS modification method conflict* occurs when multiple options exist to determine the modification method to be applied. There are two types of RQS modification method conflicts: RQS modification method generation conflicts, and RQS modification method selection conflicts.

An *RQS modification method generation conflict* occurs when multiple proposals exist for the modification method to be applied. An *RQS modification method selection conflict* occurs when there is no best proposal that meets all applicable criteria to select a modification method.

Example 14.3. Suppose that a preliminary design of the house has been made, but without an indication of the position of the front door, and suppose that there is no requirement in the current requirement qualification set regarding this position. To extend the current requirement qualification set with a front-door position requirement, one method is to ask both the customer and the architect for their preferences and come to an agreement; another method is to search for front-door position requirements in earlier design cases and choose the most prevailing one. This is an RQS modification method generation conflict.

To select one of these options, the criteria of commitment and historical dominance apply. According to the first criterion, the best choice for the modification method would be to have the customer and the architect agree upon a front-door position requirement. According to the second criterion, the best choice would be to look back in history for a prevailing front-door position requirement. This is an RQS modification method selection conflict. ■

RQS alteration conflicts

An *RQS alteration* is an intermediate or end product of modifying the contents of an RQS modification focus. Types of RQS alterations are: (1) addition of design requirement information, (2) deletion of design requirement information, and (3) modification of an existing design requirement information. An *RQS alteration conflict* occurs when multiple options exist to determine (with the current modification method) an alteration of the contents of the current modification focus. There are two types of RQS alteration conflicts: RQS alteration generation conflicts, and RQS alteration selection conflicts.

An *RQS alteration generation conflict* occurs when multiple proposals exist for the alteration to be made. An *RQS alteration selection conflict* occurs when there is no best proposal that meets all applicable criteria to select an alteration.

Example 14.4. Suppose that a preliminary design of the house has been made, but without an indication of the position of the front door. Suppose further that the modification method is to have the customer and the architect agree upon a front-door position requirement.

Given the location of the house to be built, the customer wants the front door of the house to be positioned on the west-side of the house (because then the front door can be easily accessed by visitors). However, the architect wants the front door to be on the south-side, knowing that the prevailing wind comes from the west. This is an RQS alteration generation conflict.

To select one of these options, both the criterion of accessibility for visitors and the criterion of protection against out-door conditions apply. According to the first criterion, the best option would be to require the front door to be on the west-side of the house. According to the second criterion, the best option would be to require the front door to be on the south-side. This is an RQS alteration selection conflict. ■

14.2.1.3 RQS refinement conflicts

A requirement qualification set can be refined in two ways. One way is by *deductive refinement*, that is, by adding design requirement information that is implied by (part of) the present design requirement information, given a design requirements domain theory. Another way is by *extension*, that is, by adding design requirement information that is not implied by the present design requirement information. An *RQS refinement conflict* occurs when a requirement qualification set is inconsistent with one of its refinements. There are two types of RQS refinement conflicts: deductive RQS refinement conflicts, and RQS extension conflicts.

A *deductive RQS refinement conflict* occurs when a requirement qualification set S is inconsistent with a requirement qualification set that is a deductive refinement of S. That is, a consistent design object description that fulfils S or that is indecisive with respect to fulfilment of S, fails to fulfil the deductive refinement of S. An *RQS extension conflict* occurs when a requirement qualification set S is inconsistent with an extension of S. That is, a consistent design object description that fulfils S or that is indecisive with respect to fulfilment of S, fails to fulfil the extension of S.

Example 14.5. Suppose the customer wants a two-storey house with a floor area of 50 m^2, which may cost at most 45,000 euro. For a house, a standard requirement is that a house has a floor height of 2.60 meter or more. On the basis of domain-specific knowledge that the cost of building a house is between 200 and 300 euro per m^3, it is deduced that the house will cost at least 52,000 euro. This is a deductive RQS refinement conflict. ■

14.2.2 DOD manipulation conflicts

A DOD manipulation conflict is a design conflict that is caused by a design object description manipulation process. There are three main types of DOD manipulation conflicts: DOD assessment conflicts, DOD modification conflicts and DOD refinement conflicts.

14.2.2.1 DOD assessment conflicts

A *DOD assessment conflict* concerns a specific design object description that does not fulfil a given requirement qualification set. There are two types of DOD assessment conflicts: satisfaction based DOD assessment conflicts, and fulfilment based DOD assessment conflicts. A *satisfaction based DOD assessment conflict* occurs when a specific design object description violates a specific design requirement from a given requirement qualification set. A *fulfilment based DOD assessment conflict* occurs when a specific design object description violates at least one of the qualified requirements from a given requirement qualification set.

A further distinction among DOD assessment conflicts can be made on the basis of views. A *DOD view* is a sub-set of the domain object information included in a design object description, of which the members have something in common. A commonality may be the *source* (e.g., the architect), the *nature* (e.g., fire prevention materials) or an *aspect* of the design object (e.g., the foundation of a building).

If a number of DOD views are defined for a given design object description, then two types of DOD assessment conflicts can be distinguished: intra-view DOD assessment conflicts and inter-view DOD assessment conflicts. For a specific requirement qualification set, an *intra-view DOD assessment conflict* occurs specifically within one DOD view, whereas an *inter-view DOD assessment conflict* occurs within a combination of two or more different DOD views. Similar definitions for *intra-view/inter-view satisfaction based DOD assessment conflicts* as well as *intra-view/inter-view fulfilment based DOD assessment conflicts* can be given.

Example 14.6. Suppose the customer wants the house to have a volume of about 195 m^3, but the architect has designed a house with a volume of 260 m^3. This is a satisfaction based DOD assessment conflict. If there is a DOD view for the design information about the dimensions of the house, then the conflict is an intra-view satisfaction based DOD assessment conflict. ■

14.2.2.2 DOD modification conflicts

A *DOD modification conflict* is caused by a DOD modification process. There are three types of DOD modification conflicts: DOD modification focus conflicts, DOD modification method conflicts, and DOD alteration conflicts.

DOD modification focus conflicts

A *DOD modification focus* is a sub-set of the domain object information included in a specific design object description that is to be modified (by adding new domain object information, deleting existing domain object information or changing the contents of existing domain object information). A *DOD modification focus conflict* occurs when multiple options exist to determine a new modification focus. There are two types of DOD modification focus conflicts: DOD modification focus generation conflicts, and DOD modification focus selection conflicts.

A *DOD modification focus generation conflict* occurs when multiple proposals exist for a new DOD modification focus. A *DOD modification focus selection conflict* occurs when there is no best proposal that meets all applicable criteria to select a modification focus.

Example 14.7. Suppose that in the current description of the design, the water supply system and the sewage discharge system occupy the same space in the bathroom. To resolve this conflict, three options exist: to modify the design of the routing of the water supply system, the design of the routing of the sewage discharge system, or both routings. This is a DOD modification focus generation conflict.

To select an option, the criteria of simplicity (of the re-design process) and costs apply. According to the first criterion, the best choice for the modification method would be to focus on the routing of the water supply system. According to the second criterion, the best choice would be to focus on both routings. This is a DOD modification focus selection conflict. ■

DOD modification method conflicts

A *DOD modification method* is a method for modifying the domain object information included in a design object description. Examples of DOD modification methods are (1) the refinement of parts information into detailed part parameter settings and (2) the removal of inconsistent domain object information. A *DOD modification method conflict* occurs when multiple options exist to determine the modification method to be applied. There are two types of DOD modification method conflicts: DOD modification method generation conflicts, and DOD modification method selection conflicts.

A *DOD modification method generation conflict* occurs when multiple proposals exist for the modification method to be applied. A *DOD modification method selection conflict* occurs when there is no best proposal that meets all applicable criteria to select a modification method.

Example 14.8. Suppose, as earlier, that in the current description of the design, the water supply system and the sewage discharge system occupy the same space in the bathroom. To resolve this conflict, the design of the routings of the water supply system and the sewage discharge system have to be modified. Since there is sufficient space in the bathroom, one method is to revise the routings simultaneously, whereas another method is to first re-design one routing and then the other. This is a DOD modification method generation conflict.

To select an option, the criteria of re-design process efficiency and optimality of the re-design solution apply. According to the first criterion, the best choice would be to re-design the two routings sequentially: the routing of the system that occupies the most space should be designed first. According to the second criterion, the best choice would be to re-design the routings simultaneously (so that the length of the pipes and the number of pipe bends can be minimised). This is a DOD modification method selection conflict. ■

DOD alteration conflicts

A *DOD alteration* is an intermediate or end product of modifying the contents of a DOD modification focus. Types of DOD alterations are: (1) addition of domain object information, (2) deletion of existing domain object information, and (3) modification of existing domain object information. A *DOD alteration conflict* occurs when multiple options exist to determine (with the current modification method) an alteration of the contents of the current modification focus. There are two types of DOD alteration conflicts: DOD alteration generation conflicts, and DOD alteration selection conflicts.

A *DOD alteration generation conflict* occurs when multiple proposals for the alteration to be made exist. A *DOD alteration selection conflict* occurs when there is no best proposal that meets all applicable criteria to select an alteration.

Example 14.9. Suppose the customer wants the house to have a volume of approximately 260 m^3. One possible design is that of a bungalow with an area of 100 m^2, while another possible design is that of a two-storey house with an area of 50 m^2 (in both cases assuming a floor height of 2.60 meter). This is a DOD alteration generation conflict.

To select one of these options, both the criterion of average room area and the criterion of (warmth) insulation value apply. According to the first criterion, the best choice would be the bungalow, since a bungalow's floor area can be fully occupied as room area, whereas in the two-storey house space must be reserved for stairs. According to the second criterion, the best choice would be the two-storey house, since the sum of the wall area and the roof area of the two-storey house is less than that of the bungalow. This is a DOD alteration selection conflict. ■

14.2.2.3 DOD refinement conflicts

A design object description can be refined in two ways. One way is by *deductive refinement*, that is, by adding domain object information that is implied by (part of) the present domain object information, given a design object domain theory. Another way is by *extension*, that is, by adding domain object information that is not implied by the present domain object information. A *DOD refinement conflict* occurs when a design object description is inconsistent with one of its refinements. There are two types of DOD refinement conflicts: deductive DOD refinement conflicts, and DOD extension conflicts.

A *deductive DOD refinement conflict* occurs when a design object description *D* is inconsistent with a design object description that is a deductive refinement of *D*. That is, there is specific domain object information that is either true or false in *D* and that has the opposite truth value in the deductive refinement of *D*. A *DOD extension conflict* occurs when a design object description is inconsistent with an extension of *D*. That is, there is specific domain object information that is either true or false in *D* and that has the opposite truth value in the extension of *D*.

Example 14.10. In a specific description of the design of a house, the central heating furnace is connected to an air discharge channel. On the basis of domain-specific knowledge that a furnace is always connected to a gas discharge channel, it is derived that the channel to which this specific central heating furnace is connected is a gas discharge channel. This results in a deductive DOD refinement conflict. ■

14.2.3 Design process co-ordination conflicts

For an individual design agent, there are also design conflicts that may occur within a design process co-ordination process. A *design process co-ordination conflict* occurs when the design process violates a design process objective, or when there is no clarity about the overall design strategy for the design process. There are two types of design process co-ordination conflicts: design process objective satisfaction conflicts, and design strategy conflicts.

A *design process objective satisfaction conflict* occurs when the design process violates a specific design process objective (e.g., a deadline for the design process). The occurrence of this type of conflict does not necessarily mean that the design process cannot (or, if it is still active, will not) generate a well defined requirement qualification set and/or a satisfactory design object description.

A *design strategy conflict* occurs when multiple options exist to determine a new overall design strategy. There are two types of design strategy conflicts: design strategy generation conflicts, and design strategy selection conflicts. A *design strategy generation conflict* occurs when multiple proposals for a new overall design strategy exist. A *design strategy selection conflict* occurs when there is no best proposal that meets all applicable criteria to select an overall design strategy.

Example 14.11. In an early stage of a specific design process, the architect knows that she will be unsuccessful in making a design of a house that satisfies the customer's requirements, because some of these design requirements are in conflict with each other. She could first negotiate the design requirements with the customer (e.g., relax or replace some of the conflicting requirements) and then make a satisfactory design. Alternatively, she could first make a partial design that satisfies the non-conflicting design requirements, then negotiate the conflicting design requirements with the customer, and finally finish the design. This is a design strategy generation conflict.

To select one of these options, the criteria of early customer involvement and late customer involvement apply. According to the first criterion, the best choice would be to negotiate the requirements with the customer first. According to the second criterion, the best choice would be to make a partial design first. This is a design strategy selection conflict. ■

14.2.4 Multi-agent design process conflicts

In many domains, it is not uncommon to have multiple design agents work together on the same design problem. In a car company, for example, industrial designers, electric engineers, interface designers and mechanical engineers may co-operate to design a new car. In a design team, each participant contributes design requirements and/or part of the design object description. A *co-operation conflict* occurs when two or more design agents cannot agree about issues that are relevant for effective and efficient co-operation. There are five types of co-operation conflicts: language conflicts, belief conflicts, goal conflicts, responsibility conflicts, and co-operation level conflicts.

A *language conflict* occurs when the design agents use incompatible (natural or technical) languages to communicate with each other. A *belief conflict* occurs when the design agents have incompatible sets of beliefs; this may happen when the design agents are from different fields of expertise or different schools or have a different cultural background. A *goal conflict* occurs when the design agents have incompatible goals; this may happen when leadership is lacking within the design team. A *responsibility conflict* occurs when the design agents have overlapping or insufficient responsibilities; this may happen as a result of a poor delegation of roles within the design team. A *co-operation level conflict* occurs when the design agents disagree about the co-operation mechanisms such as the means and methods for the generation, approval and integration of sets of design requirements and designs.

Example 14.12. A team of industrial designers convenes to develop a new commercial product. ShoeString, a company that is the result of a recent merger of the leading design companies String Product Innovation and W.H. Shoe, employs them all. During the lively meeting, the designers appear to have totally different perceptions about what the new product should offer to customers. This is a goal conflict.

After having agreed upon the requirements and main components of the new product, the designers split up in task groups to work out the details of the product. The designers formerly from W.H. Shoe are used to using a PC-based CAD system, whereas the designers formerly from String Product Innovation are used to drafting by hand. Each method requires its own vocabulary, causing a language conflict.

The designers agree to make sketches by hand first and then to use the CAD system to produce a three-dimensional image. Later, during a presentation of the results of each of the task groups, it appears that two groups have been working on the same component of the new product. This is a responsibility conflict.

After having resolved the responsibility issue, the designers sit together to decide about a joint proposal for the new product, based on the results produced by the task groups. However, the designers cannot agree about a voting method for approval of these results: some argue that they should agree unanimously, others feel that a normal majority suffices. This is a co-operation level conflict. ■

14.3 Management of RQS Manipulation Conflicts

The generic model of design, GDM, and its specialisations described in Chapter 9 have been used to model and specify the management of the different types of RQS manipulation conflicts described in Section 14.2.1. RQS manipulation conflicts are managed primarily by the RQS manipulation process of a design process. However, it is possible that for the detection of RQS assessment conflicts, the DOD manipulation process is involved as well.

RQS manipulation conflict detection. The following enumeration shows how each type of RQS manipulation conflict is detected and by which design sub-process.

- *RQS assessment conflicts.* If RQS assessment or DOD assessment concludes that it is impossible to generate a satisfactory design object description for a given requirement qualification set, then an RQS assessment conflict occurs. This conflict may be due to a lack of knowledge about how to fulfil the given requirement qualification set, which is apparent by the presence of qualified requirements included in the given requirement qualification set that cannot be decided to be satisfiable or not. The conflict may also be due to an inconsistency in the given requirement qualification set, which is apparent by the presence of qualified requirements included in the given requirement qualification set, of which always at least one is violated by a design object description. If RQS views for the given requirement qualification set exist, then there are two cases. If the RQS assessment conflict occurs within a specific RQS view, then this conflict is more specifically an intra-view RQS assessment conflict. If the conflict occurs not within one RQS view, but within the union of specific RQS views, then this conflict is more specifically an inter-view RQS assessment conflict.

- *RQS modification focus conflicts.* If RQS modification focus determination generates multiple proposals for the new RQS modification focus on the current requirement qualification set, then an RQS modification focus generation conflict occurs. Subsequently, if RQS modification focus determination applies criteria to select an RQS modification focus from the proposals, and the number of proposals that meet all of these criteria does not equal one, then an RQS modification focus selection conflict occurs.

- *RQS modification method conflicts.* If RQS modification method determination generates multiple proposals for the RQS modification method to be used to modify the current requirement qualification set, then an RQS modification method generation conflict occurs. Subsequently, if RQS modification method determination applies criteria to select an RQS modification method from the proposals, and the number of proposals that meet all of these criteria does not equal one, then an RQS modification method selection conflict occurs.

- *RQS alteration conflicts.* If RQS modification method execution generates multiple proposals for the next RQS alteration (i.e., set of modifications to be applied) to the current requirement qualification set, then an RQS alteration generation conflict occurs. Subsequently, if RQS modification method execution applies criteria to select an RQS alteration from the proposals, and the number of proposals that meet all of these criteria does not equal one, then an RQS alteration selection conflict occurs.

- *RQS refinement conflicts.* If RQSM process evaluation concludes that the current requirement qualification set is inconsistent with an earlier generated requirement qualification set, and the current requirement qualification set has been formed by adding design requirement information to the earlier requirement qualification set, then an RQS refinement conflict occurs. If the earlier requirement qualification set has been deductively refined to form the current requirement qualification set, then the conflict is more specifically a deductive RQS refinement conflict. Otherwise, if the earlier requirement qualification set has been extended to form the current requirement qualification set, then the conflict is more specifically an RQS extension conflict.

RQS manipulation conflict prioritisation. Some types of RQS manipulation conflicts are resolved on a *Last In First Out* basis, which means that attempts to resolve a conflict start when the conflict is detected. This is the case for (1) RQS modification focus conflicts, which are solved by RQS modification focus determination, (2) RQS modification method conflicts, which are solved by RQS modification method determination, and (3) RQS alteration conflicts, which are solved by RQS modification method execution. But the *Last In First Out* rule does not necessarily apply for other types of RQS manipulation conflicts.

For domain-specific reasons or for reasons of efficiency of the resolution process, it may be the case that RQS modification focus determination selects a focus that includes only part of the known RQS assessment conflicts and RQS refinement conflicts. For example, there may be knowledge in RQS modification focus determination which assigns a priority to a conflict on the basis of whether the conflict involves design requirements that are inconsistent, imprecise, or ambiguous (e.g., an inconsistent design requirement may be more important to be resolved than an ambiguous design requirement). If the selected focus does not include the RQS assessment conflict or the RQS refinement conflict that has just been detected, then the resolution of this conflict is effectively postponed. This corresponds to design practice, where it is not unusual that the designer focuses on a few conflicts right away and leaves the rest for later.

RQS manipulation conflict resolution. The following enumeration shows how each type of RQS manipulation conflict is resolved and by which design sub-process.

- *RQS assessment conflicts.* RQS modification determination resolves an RQS assessment conflict by selecting an alteration to the current requirement qualification set

that deletes and/or changes design requirements involved in the conflict. (During this process, it may happen that an RQS modification conflict, an RQS modification method conflict, or an RQS alteration conflict occurs.)

- *RQS modification focus conflicts.* RQS modification focus determination resolves an RQS modification focus generation conflict by applying criteria for the selection of one of the RQS modification focus proposals. It may happen that an RQS modification focus selection conflict occurs, which is resolved by selecting an RQS modification focus proposal with the largest sum of weights of supporting criteria.

- *RQS modification method conflicts.* RQS modification method determination resolves an RQS modification method generation conflict by applying criteria for the selection of one of the RQS modification method proposals. It may happen that an RQS modification method selection conflict occurs, which is resolved by selecting an RQS modification method proposal with the largest sum of weights of supporting criteria.

- *RQS alteration conflicts.* RQS modification method execution resolves an RQS alteration generation conflict by applying criteria for the selection of one of the RQS alteration proposals. It may happen that an RQS alteration selection conflict occurs, which is resolved by selecting an RQS alteration proposal with the largest sum of weights of supporting criteria.

- *RQS refinement conflicts.* RQS modification determination resolves an RQS refinement conflict by selecting an alteration to the current requirement qualification set that deletes and/or changes design requirements involved in the conflict. (During this process, it may happen that an RQS modification conflict, an RQS modification method conflict or an RQS alteration conflict occurs.)

In the following sub-sections, it is shown how the management of RQS manipulation conflicts can be specified in accordance with the model described above. For the purpose of illustration, examples from Section 14.2.1 are used.

14.3.1 Management of RQS assessment conflicts

Example 14.1 presents an RQS fulfilment conflict concerning the provision of water in a garage. The customer wants a tap but no central heating in the garage, whereas the architect insists on central heating in a garage that has a tap. The management of this conflict is modelled as follows.

Detection of RQS assessment conflicts. In Example 14.1, the design requirements that form the basis of the conflict are three requirements and two qualified requirements. These design requirements are modelled as follows.

Requirement R1a states that the garage must have a tap, requirement R1b states that the garage may not have central heating, and qualified requirement QR1a (defined by the customer) states that the requirements R1a and R1b must both be satisfied:

```
is-defined-as(R1a,
  for-all(L, l-implies(has-value(garage1, infrastructure, L),
      exists(T, l-and(is-member-of(T, L), has-value(T, type, tap))))))
is-defined-as(R1b,
  for-all(L, l-implies(has-value(garage1, infrastructure, L),
      l-not(exists(T, l-and(is-member-of(T, L), has-value(T, type, central-heating)))))))
is-defined-as(QR1a, every, [R1a, R1b])
is-source-of(customer, QR1a)
is-to-be-satisfied(QR1a)
```

Requirement R1c states that if the garage has a tap, then it must also have central heating, and qualified requirement QR1b (defined by the architect) states that the requirement R1c must be satisfied:

```
is-defined-as(R1c,
  for-all(L, for-all(T,
    l-implies(l-and(has-value(garage1, infrastructure, L),
        l-and(is-member-of(T, L), has-value(T, type, tap))),
      exists(E, l-and(is-member-of(E, L), has-value(E, type, central-heating)))))))
is-defined-as(QR1b, every, [R1c])
is-source-of(architect, QR1b)
is-to-be-satisfied(QR1b)
```

After activating the DOD manipulation process to reason with this set of design requirements, an RQS assessment conflict occurs, as DOD assessment notices that a design object description that satisfies qualified requirement QR1a necessarily violates qualified requirement QR1b, and vice versa. Therefore, it is concluded that the qualified requirements QR1a and QR1b are inconsistent.

After having assigned the highest priority to this single RQS assessment conflict, the next step for RQS manipulation is to resolve the RQS assessment conflict by modifying the given requirement qualification set.

Resolution of RQS assessment conflicts. As the modification of requirement qualification sets is discussed extensively in the following sub-section, the RQS modification process to

resolve the RQS assessment conflict in Example 14.1 is described briefly and informally. First of all, the RQS modification focus is set on the qualified requirements involved in the conflict, QR1a and QR1b.

Then the RQS modification method chosen is to replace (one of) the design requirements in focus by new design requirements. One of the two alteration proposals, put forward by the customer, is to drop the requirement concerning the tap, R1a:

```
is-proposed-RQS-alteration(RQS-alteration1a)
includes-RQS-modification(RQS-alteration1a, deletion-of(is-to-be-satisfied(QR1a)))
includes-RQS-modification(RQS-alteration1a, addition-of(is-defined-as(QR1c, every, [R1b])))
includes-RQS-modification(RQS-alteration1a, addition-of(is-source-of(customer, QR1c)))
includes-RQS-modification(RQS-alteration1a, addition-of(is-to-be-satisfied(QR1c)))
```

A second alteration proposal put forward by the architect is to replace requirement R1c by one that defines electrical heating as the source of heating for a garage with a tap:

```
is-proposed-RQS-alteration(RQS-alteration1b)
includes-RQS-modification(RQS-alteration1b, deletion-of(is-to-be-satisfied(QR1b)))
includes-RQS-modification(RQS-alteration1b, addition-of(is-defined-as(R1d,
  for-all(L, for-all(T,
    l-implies(l-and(has-value(garage1, infrastructure, L),
        l-and(is-member-of(T, L), has-value(T, type, tap))),
      exists(E, l-and(is-member-of(E, L), has-value(E, type, electric-heating)))))))))
includes-RQS-modification(RQS-alteration1b, addition-of(is-defined-as(QR1d, every, [R1d])))
includes-RQS-modification(RQS-alteration1b, addition-of(is-source-of(architect, QR1d)))
includes-RQS-modification(RQS-alteration1b, addition-of(is-to-be-satisfied(QR1d)))
```

To select one of the two alteration proposals, the customer and architect agree to apply a cost-minimisation criterion. With this criterion, the proposal to drop the tap requirement is selected, because it is cheaper not to install a tap than to install a tap and a heating device.

14.3.2 Management of RQS modification conflicts

The management of RQS modification conflicts follows a distinct pattern. In the beginning of Section 14.3, it has been explained that the ways in which RQS modification focus conflicts, RQS modification method conflicts, and RQS alteration conflicts are detected, prioritised, and resolved are comparable. Therefore, rather than elaborating the management of all different types of RQS modification conflicts, this sub-section presents specifications of knowledge for managing one type of RQS modification conflict: RQS modification focus conflicts.

Example 14.2 presents an RQS modification focus generation conflict that occurs in an early stage of designing a house. One option is to concentrate on the non-conflicting design requirements of the dormer, as the customer suggests, and another option is to focus on the conflicting design requirements of the living room, as the architect suggests.

During the selection of one of these options, an RQS modification focus selection conflict occurs. The criterion of flexibility applies, sustaining the customer's suggestion, as well as the criterion of efficiency of the design process, sustaining the architect's suggestion. The management of these two RQS modification focus conflicts is modelled as follows.

Detection of RQS modification focus conflicts. The RQS modification focus generation conflict in Example 14.2 is detected as follows. The design parties involved in the RQS manipulation process, the customer and the architect, are given the opportunity to propose new RQS modification foci. This process is modelled by the component RQS-modification-focus-generation, which is a sub-component of the component RQS-modification-focus-determination.

In Example 14.2, there is a proposal by the customer to concentrate on the design requirements of the dormer, and a proposal by the architect to focus on the design requirements of the living room. This information is modelled as part of the output of the component RQS-modification-focus-generation:

```
is-proposed-RQS-modification-focus(design-reqs(dormer1))
is-source-of-RQS-modification-focus-proposal(customer, design-reqs(dormer1))
is-proposed-RQS-modification-focus(design-reqs(living-room1))
is-source-of-RQS-modification-focus-proposal(architect, design-reqs(living-room1))
```

The RQS modification focus generation conflict is caused by the presence of multiple proposals for a new RQS modification focus.

The RQS modification focus selection conflict in Example 14.2 is detected as follows. Available domain-specific knowledge is that there are two applicable criteria: flexibility and efficiency. A modification focus meets the criterion of flexibility if the customer has proposed it, and it meets the criterion of efficiency if it contains conflicting design requirements. Available task-specific knowledge is that the best RQS modification focus is one that meets all applicable criteria. Together, this application-specific knowledge is modelled by the following facts and rules within the knowledge base of the component RQS-modification-focus-proposal-evaluation, which is a sub-component of the component RQS-modification-focus-determination:

```
is-set-of-applicable-RQS-modification-focus-selection-criteria([flexibility, efficiency]);

if is-source-of(customer, F: RQS-modification-focus)
then meets-criterion(F: RQS-modification-focus, flexibility);
```

```
if is-source-of(D: design-party, F: RQS-modification-focus)
  and not D: design-party = customer
then not meets-criterion(F: RQS-modification-focus, flexibility);

if contains-conflicting-design-requirements(F: RQS-modification-focus)
then meets-criterion(F: RQS-modification-focus, efficiency);

if not contains-conflicting-design-requirements(F: RQS-modification-focus)
then not meets-criterion(F: RQS-modification-focus, efficiency);

meets-all-criteria-from(F: RQS-modification-focus, [ ]);

if meets-all-criteria-from(F: RQS-modification-focus, L: criterion-list)
  and meets-criterion(F: RQS-modification-focus, C: criterion)
then meets-all-criteria-from(F: RQS-modification-focus, [C: criterion | L: criterion-list]);

if is-set-of-applicable-RQS-modification-focus-selection-criteria(L: criterion-list)
  and meets-all-criteria-from(F: RQS-modification-focus, L: criterion-list)
then is-best-RQS-modification-focus(F: RQS-modification-focus);
```

If, in Example 14.2, the component RQS-modification-focus-proposal-evaluation is activated to reason with the information that the customer has proposed to modify the (non-conflicting) design requirements of the dormer, and that the architect has proposed to modify the conflicting design requirements of the living room:

```
is-source-of-RQS-modification-focus(customer, design-reqs(dormer1))
not contains-conflicting-design-requirements(design-reqs(dormer1))
is-source-of-RQS-modification-focus(architect, design-reqs(living-room))
contains-conflicting-design-requirements(design-reqs(living-room1))
```

and if the decision targets of this component are to confirm that the two given RQS modification focus proposals are the best:

```
target(decision, is-best-RQS-modification-focus(design-reqs(dormer1)), confirm)
target(decision, is-best-RQS-modification-focus(design-reqs(living-room1)), confirm)
```

then its output includes the information that the criteria of flexibility and efficiency apply, and that the RQS modification focus proposals each meet a different criterion:

```
is-set-of-applicable-RQS-modification-focus-selection-criteria([flexibility, efficiency])
meets-criterion(design-reqs(dormer1), flexibility)
not meets-criterion(design-reqs(dormer1), efficiency)
not meets-criterion(design-reqs(living-room1), flexibility)
meets-criterion(design-reqs(living-room1), efficiency)
```

The RQS modification focus selection conflict is caused by the absence of a best proposal, since neither one of the RQS modification focus proposals meets both applicable criteria. The conflict is modelled by task control information that the component RQS-modification-focus-proposal-evaluation has failed to confirm any of its decision targets.

Resolution of RQS modification focus conflicts. The RQS modification focus conflicts in Example 14.2 are resolved in three steps: weighing of RQS modification focus proposals on the basis of applicable criteria, ranking of RQS modification focus proposals on the basis of their weights, and selection of a best RQS modification focus proposal.

Firstly, weights are assigned to the applicable criteria, reflecting their importance in the selection of an RQS modification focus proposal, given the current stage of the design process. Available domain-specific knowledge is that three design process stages exist: early, middle, and late. Flexibility is more preferred in an earlier stage, as it may be more difficult to adhere to new design requirements in a later stage. Efficiency is more preferred in a later stage, since it is less important to concentrate on removing conflicts among design requirements in an earlier stage.

Available task-specific knowledge is that the weight of a proposed RQS modification focus equals the sum of weights of the criteria that it meets. Together, this application-specific knowledge is modelled by the following facts and rules within the knowledge base of the component RQS-modification-focus-proposal-weighing, which is a sub-component of the component RQS-modification-focus-determination:

```
;; Weights are assigned to criteria on the basis of the stage of the design process.

has-weight-in-design-stage(flexibility, early-stage, 1.0);
has-weight-in-design-stage(flexibility, middle-stage, 0.5);
has-weight-in-design-stage(flexibility, late-stage, 0.0);
has-weight-in-design-stage(efficiency, early-stage, 0.0);
has-weight-in-design-stage(efficiency, middle-stage, 0.5);
has-weight-in-design-stage(efficiency, late-stage, 1.0);

is-member-of(C: criterion, [C: criterion | L: criterion-list]);

if is-member-of(C1: criterion, L: criterion-list)
then is-member-of(C1: criterion, [C2: criterion | L: criterion-list]);
```

```
if is-set-of-applicable-RQS-modification-focus-selection-criteria(L: criterion-list)
  and is-member-of(C: criterion, L: criterion-list)
  and is-current-design-stage(DPS: design-process-stage)
  and has-weight-in-design-stage(C: criterion, DPS: design-process-stage, W: real)
then has-weight(C: criterion, W: real);

;; Weights are assigned to proposed foci on the basis of the criteria that they meet.

has-summed-weight(F: RQS-modification-focus, [ ], 0.0);

if has-summed-weight(F: RQS-modification-focus, L: criterion-list, OldW: real)
  and meets-criterion(F: RQS-modification-focus, C: criterion)
  and has-weight(C: criterion, CriterionW: real)
  and W: real = OldW: real + CriterionW: real
then has-summed-weight(F: RQS-modification-focus, [C: criterion | L: criterion-list], W: real);

if has-summed-weight(F: RQS-modification-focus, L: criterion-list, W: real)
  and not meets-criterion(F: RQS-modification-focus, C: criterion)
then has-summed-weight(F: RQS-modification-focus, [C: criterion | L: criterion-list], W: real);

if is-proposed-RQS-modification-focus(F: RQS-modification-focus)
  and is-set-of-applicable-RQS-modification-focus-selection-criteria(L: criterion-list)
  and has-summed-weight(F: RQS-modification-focus, L: criterion-list, W: real)
then has-weight(F: RQS-modification-focus, W: real);
```

If, in Example 14.2, the component RQS-modification-focus-proposal-weighing is activated to reason with the following information: (1) the design process is in an early stage, (2) there is one proposal to focus on the design requirements of the dormer and one proposal to focus on the design requirements of the living room, (3) the criteria of flexibility and efficiency apply, and (4) the proposal to focus on the design requirements of the dormer meets flexibility but not efficiency, and the proposal to focus on the design requirements of the living room meets efficiency but not flexibility:

```
is-current-design-stage(early-stage)

is-proposed-RQS-modification-focus(design-reqs(dormer1))
is-proposed-RQS-modification-focus(design-reqs(living-room1))

is-set-of-applicable-RQS-modification-focus-selection-criteria([flexibility, efficiency])
```

```
meets-criterion(design-reqs(dormer1), flexibility)
not meets-criterion(design-reqs(dormer1), efficiency)
not meets-criterion(design-reqs(living-room1), flexibility)
meets-criterion(design-reqs(living-room1), efficiency)
```

then its output includes the information that the proposal to focus on the design requirements of the dormer has a weight of 1.0 and the proposal to focus on the design requirements of the living room has a weight of 0.0:

```
has-weight(design-reqs(dormer1), 1.0)
has-weight(design-reqs(living-room1), 0.0)
```

Secondly, the RQS modification focus proposals are ranked on the basis of their weights, in order to determine the best RQS modification focus: a heavier weight results in a higher rank, equal weights in the same rank. This task-specific knowledge is modelled by facts and rules within the knowledge base of the component RQS-modification-focus-proposal-ranking, of which the details are omitted here (because of the large size of the specification). In Example 14.2, the best RQS modification focus is the set of design requirements of the dormer.

Thirdly, a proposed RQS modification focus is selected that has been determined to be the best. This task-specific knowledge is modelled by a rule within the knowledge base of the component RQS-modification-focus-proposal-selection, which is a sub-component of the component RQS-modification-focus-determination:

```
if is-best-RQS-modification-focus(F: RQS-modification-focus)
then is-selected-RQS-modification-focus(F: RQS-modification-focus);
```

The targets of this component are to confirm the selection of proposed RQS modification foci. It may happen that there is more than one best focus, in which case one of them has to be selected at random. This knowledge is specified by the initial evaluation criterion any of the component RQS-modification-focus-proposal-selection, stating that the component may terminate its activation when it has achieved any of its targets.

In Example 14.2, the set of design requirements of the dormer is selected as the new RQS modification focus, which ends the resolution of the two RQS modification focus conflicts.

14.3.3 Management of RQS refinement conflicts

Example 14.5 presents a deductive RQS refinement conflict concerning a house required by the customer to have two floors, each with an area of 50 m^2, and to cost at most 45,000 euro. After deductive refinement of the customer's design requirements the costs are calculated to be at least 52,000 euro. The management of this conflict is modelled as follows.

Detection of RQS refinement conflicts. In Example 14.5, the design requirements that form the basis of the deductive RQS refinement conflict are five requirements and one qualified requirement. These design requirements are modelled as follows.

Requirement R5a states that the design object must be a house, requirement R5b states that the house must have two floors, requirements R5c and R5d state that these floors must each have an area of 50 m^2, requirement R5e states that the building costs of the house may not exceed 45,000 euro, and qualified requirement QR5a states that the requirements R5a to R5e must all be satisfied:

```
is-defined-as(R5a, has-value(house1, type, house))
is-defined-as(R5b, has-value(house1, floors, [floor1, floor2]))

is-defined-as(R5c, has-value(floor1, area, val(50, m^2))
is-defined-as(R5d, has-value(floor2, area, val(50, m^2))

is-defined-as(R5e,
  for-all(N, l-implies(has-value(house1, building-costs, val(N, EUR)), le(N, 45000))))

is-defined-as(QR5a, every, [R5a, R5b, R5c, R5d, R5e])

is-to-be-satisfied(QR5a)
```

The RQS manipulation process deductively refines this set of design requirements. Available domain-specific knowledge is that:

- if the design object is to be a house, then there is an official building requirement stating that the height of each floor must be at least 2.60 meter,
- the required volume of a floor equals the product of the required area of the floor and the required height of the floor,
- the required volume of a house equals the sum of the required volumes of the house's floors, and
- the building costs of a house equal the product of the house's required volume and the building costs per volume unit; the minimum building costs are 200 euro per m^3, and the maximum building costs are 300 euro per m^3.

This domain-specific knowledge is modelled by the following facts and rules within the knowledge base of the component deductive-RQS-refinement:

```
is-member-of(O: domain-object, [O: domain-object | L: domain-object-list]);

if is-member-of(O1: domain-object, L: domain-object-list)
then is-member-of(O1: domain-object, [O2: domain-object | L: domain-object-list]);

;; Every floor of a house must have a height of at least 2.60 meter.

if is-defined-as(R1: requirement-name, has-value(O: domain-object, type, house))
  and is-defined-as(R2: requirement-name, has-value(O: domain-object, floors, L: floor-list)
  and is-member-of(F: floor, L: floor-list)
then is-defined-as(standard-req(object-attribute(F: floor, minimum-height)),
    for-all(N, l-implies(has-value(F: floor, height, val(N, m)), ge(N, 2.60))));

;; The required volume of a floor equals the product of its required area and its required height.

if is-defined-as(R1: requirement-name,
  for-all(V: domain-variable, l-implies(has-value(F: floor, height, val(V: domain-variable, U: unit)),
      ge(V: domain-variable, N1: number))))
  and is-defined-as(R2: requirement-name, has-value(F: floor, area, val(N2: number, (U: unit)^2)))
  and N3: number = N1: number * N2: number
then is-defined-as(standard-req(object-attribute(F: floor, minimum-volume)),
    for-all(N, l-implies(has-value(F: floor, volume, val(N, (U: unit)^3)), ge(N, N3: number))));

;; The required volume of a house equals the sum of required volumes of the house's floors.

has-sum-of-required-minimum-attribute-values([], A: attribute, val(0, U: unit));

if has-sum-of-required-minimum-attribute-values(
    L: domain-object-list, A: attribute, val(V1: number, U: unit))
  and is-defined-as(R: requirement-name, for-all(V: domain-variable,
    l-implies(has-value(O: domain-object, A: attribute, val(V: domain-variable, U: unit)),
      ge(V: domain-variable, V2: number))))
  and V3: number = V1: number + V2: number
then has-sum-of-required-minimum-attribute-values(
  [O: domain-object | L: domain-object-list], A: attribute, val(V3: number, U: unit));

if is-defined-as(R1: requirement-name, has-value(O: domain-object, type, house))
  and is-defined-as(R2: requirement-name, has-value(O: domain-object, floors, L: floor-list)
  and has-sum-of-required-minimum-attribute-values(L: floor-list, volume, val(V: number, U: unit))
then is-defined-as(standard-req(object-attribute(O: domain-object, minimum-volume)),
    for-all(N, l-implies(has-value(O: domain-object, volume, val(N, U: unit)), ge(N, V: number))));
```

```
;; The building costs of a house equal the product of the house's required volume and the
;; building costs per volume unit. The minimum building costs are 200 euro per cubic meter,
;; and the maximum building costs are 300 euro per cubic meter.

if is-defined-as(R1: requirement-name, has-value(O: domain-object, type, house))
  and is-defined-as(R2: requirement-name, for-all(V: domain-variable,
      l-implies(has-value(O: domain-object, volume, val(V: domain-variable, m^3)),
        ge(V: domain-variable, N1: number))))
  and N2: number = N1: number * 200
then is-defined-as(standard-req(object-attribute(O: domain-object, minimum-building-costs),
  for-all(N, l-implies(has-value(O: domain-object, building-costs, val(N, EUR)),
    ge(N, N2: number))));

if is-defined-as(R1: requirement-name, has-value(O: domain-object, type, house))
  and is-defined-as(R2: requirement-name, for-all(V: domain-variable,
      l-implies(has-value(O: domain-object, volume, val(V: domain-variable, m^3)),
        ge(V: domain-variable, N1: number))))
  and N2: number = N1: number * 300
then is-defined-as(standard-req(object-attribute(O: domain-object, maximum-building-costs),
  for-all(N, l-implies(has-value(O: domain-object, building-costs, val(N, EUR)),
    le(N, N2: number))));

if is-defined-as(standard-req(R: requirement-name, E: domain-object-info-expression)
then is-defined-as(standard-qualified-req(R: requirement-name), every, [R: requirement-name]);

if is-defined-as(standard-req(R: requirement-name, E: domain-object-info-expression)
then is-to-be-satisfied(standard-qualified-req(R: requirement-name));
```

If, in Example 14.5, the component deductive-RQS-refinement is activated to reason with the design requirements R5a to R5d and QR5a as input information, then its output includes the new design requirement information that (1) the height of a floor must be 2.60 meter or more, (2) the floors each must have a volume of at least 130 m^3, (3) the house must have a volume of at least 260 m^3, and (4) the minimum (maximum) building costs of the house must be at least (at most) 52,000 euro (78,000 euro):

```
is-defined-as(standard-req(object-attribute(floor1, minimum-height)),
   for-all(N, l-implies(has-value(floor1, height, val(N, m)), ge(N, 2.60))))
is-defined-as(standard-req(object-attribute(floor2, minimum-height)),
   for-all(N, l-implies(has-value(floor2, height, val(N, m)), ge(N, 2.60))))
```

```
is-defined-as(standard-req(object-attribute(floor1, minimum-volume)),
   for-all(N, l-implies(has-value(floor1, volume, val(N, m^3)), ge(N, 130))))
is-defined-as(standard-req(object-attribute(floor2, minimum-volume)),
   for-all(N, l-implies(has-value(floor2, volume, val(N, m^3)), ge(N, 130))))

is-defined-as(standard-req(object-attribute(house1, minimum-volume)),
   for-all(N, l-implies(has-value(house1, volume, val(N, m^3)), ge(N, 260))))

is-defined-as(standard-req(object-attribute(house1, minimum-building-costs),
  for-all(N, l-implies(has-value(house1, building-costs, val(N, EUR)), ge(N, 52000))))
is-defined-as(standard-req(object-attribute(house1, maximum-building-costs),
  for-all(N, l-implies(has-value(house1, building-costs, val(N, EUR)), le(N, 78000))))

is-defined-as(standard-qualified-req(object-attribute(floor1, minimum-height)),
  every, [standard-req(object-attribute(floor1, minimum-height))])
is-defined-as(standard-qualified-req(object-attribute(floor2, minimum-height)),
  every, [standard-req(object-attribute(floor2, minimum-height))])

is-defined-as(standard-qualified-req(object-attribute(floor1, minimum-volume)),
  every, [standard-req(object-attribute(floor1, minimum-volume))])
is-defined-as(standard-qualified-req(object-attribute(floor2, minimum-volume)),
  every, [standard-req(object-attribute(floor2, minimum-volume))])

is-defined-as(standard-qualified-req(object-attribute(house1, minimum-volume)),
  every, [standard-req(object-attribute(house1, minimum-volume))])

is-defined-as(standard-qualified-req(object-attribute(house1, minimum-building-costs)),
  every, [standard-req(object-attribute(house1, minimum-building-costs))])
is-defined-as(standard-qualified-req(object-attribute(house1, maximum-building-costs)),
  every, [standard-req(object-attribute(house1, maximum-building-costs))])

is-to-be-satisfied(standard-qualified-req(object-attribute(floor1, minimum-height)))
is-to-be-satisfied(standard-qualified-req(object-attribute(floor2, minimum-height)))

is-to-be-satisfied(standard-qualified-req(object-attribute(floor1, minimum-volume)))
is-to-be-satisfied(standard-qualified-req(object-attribute(floor2, minimum-volume)))

is-to-be-satisfied(standard-qualified-req(object-attribute(house1, minimum-volume)))

is-to-be-satisfied(standard-qualified-req(object-attribute(house1, minimum-building-costs)))
is-to-be-satisfied(standard-qualified-req(object-attribute(house1, maximum-building-costs)))
```

If, in Example 14.5, the component RQS-assessment is activated to reason with the new requirement qualification set, it concludes that, although no conflicts have been detected in the previous requirement qualification set, this new requirement qualification set is inconsistent: the building costs of the house must be at least 52,000 euro, and at the same time the building costs must be at most 45,000 euro.

The deductive RQS refinement conflict (detected by the RQS assessment process) is caused by the presence of two contradicting requirements on the building costs of the house, and one of these requirements is the result of deductive refinement.

After having assigned the highest priority to this single RQS refinement conflict, the next step for RQS manipulation is to resolve the RQS refinement conflict by modifying the given requirement qualification set.

Resolution of RQS refinement conflicts. The conflict in Example 14.5 is resolved as follows. (As the modification of requirement qualification sets has been discussed extensively in the previous sub-section, the RQS modification process for Example 14.5 is described briefly and informally.) First of all, domain-specific knowledge is used to determine a suitable RQS modification focus on the given requirement qualification set. The proposed RQS modification foci each involve one of the following requirements:

a. requirement R5a on the building type,
b. requirement R5b on the number of floors,
c. requirement R5c on the area of the first floor,
d. requirement R5d on the area of the second floor,
e. requirement R5e on the maximum building costs.

As the proposed foci contain the customer's design requirements, the customer is asked to formulate preferences between these proposals, resulting in the selection of the proposal to focus on requirement R5e and on QR5a, the only qualified requirement included in the set.

Knowing that activation of the deductive RQS refinement process has resulted in the new design requirement that the building costs must be at least 52,000 euro, a simple method to resolve the conflict is to remove the design requirements that the building costs must be at most 45,000 euro. Hence, the selected alteration is the following:

```
is-selected-RQS-alteration(RQS-alteration5)
includes-RQS-modification(RQS-alteration5, deletion-of(is-to-be-satisfied(QR5a)))
includes-RQS-modification(RQS-alteration5,
  addition-of(is-defined-as(QR5b, every, [R5a, R5b, R5c, R5d])))
includes-RQS-modification(RQS-alteration5, addition-of(is-to-be-satisfied(QR5b)))
```

After modification of the current requirement qualification set, activation of RQS assessment results in the conclusion that there are no RQS manipulation conflicts anymore.

14.4 Management of DOD Manipulation Conflicts

The generic model of design, GDM, and its specialisations described in Chapter 9 have been used to model and specify the management of the different types of DOD manipulation conflicts described in Section 14.2.2. DOD manipulation conflicts are managed entirely by the DOD manipulation process of a design process.

DOD manipulation conflict detection. The following enumeration shows how each type of DOD manipulation conflict is detected and by which design sub-process.

- *DOD assessment conflicts.* If DOD assessment concludes that the current design object description fails to fulfil the given requirement qualification set, then a DOD assessment conflict occurs. If DOD views for the given requirement qualification set exist, then there are two cases. If the DOD assessment conflict occurs within a specific DOD view, then this conflict is more specifically an intra-view DOD assessment conflict. If the conflict occurs not within one DOD view, but within the union of specific DOD views, then this conflict is more specifically an inter-view DOD assessment conflict.

- *DOD modification focus conflicts.* If DOD modification focus determination generates more than one proposal for the new DOD modification focus on the current design object description, a DOD modification focus generation conflict occurs. Subsequently, if DOD modification focus determination applies criteria to select a DOD modification focus from the proposals, and the number of proposals that meet all of these criteria does not equal one, then a DOD modification focus selection conflict occurs.

- *DOD modification method conflicts.* If DOD modification method determination generates multiple proposals for the DOD modification method to be used to modify the current design object description, then a DOD modification method generation conflict occurs. Subsequently, if DOD modification method determination applies criteria to select a DOD modification method from the proposals, and the number of proposals that meet all of these criteria does not equal one, then a DOD modification method selection conflict occurs.

- *DOD alteration conflicts.* If DOD modification method execution generates multiple proposals for the next DOD alteration (i.e., set of modifications to be applied) to the current design object description, then a DOD alteration generation conflict occurs. Subsequently, if DOD modification method execution applies criteria to select a DOD alteration from the proposals, and the number of proposals that meet all of these criteria does not equal one, then a DOD alteration selection conflict occurs.

- *DOD refinement conflicts.* If DODM process evaluation concludes that the current design object description is inconsistent with an earlier generated design object description, and the current design object description has been formed by adding domain object information to the earlier design object description, then a DOD refinement conflict occurs. If the earlier design object description has been deductively refined to form the current design object description, then the conflict is more specifically a deductive DOD refinement conflict. Otherwise, if the earlier design object description has been altered to form the current design object description, then the conflict is more specifically a DOD extension conflict.

DOD manipulation conflict prioritisation. Some types of DOD manipulation conflicts are resolved on a *Last In First Out* basis, which means that attempts to resolve a conflict start when the conflict is detected. This is the case for (1) DOD modification focus conflicts, which are immediately solved by DOD modification focus determination, (2) DOD modification method conflicts, which are immediately solved by DOD modification method determination, and (3) DOD alteration conflicts, which are immediately solved by DOD modification method execution. But the *Last In First Out* rule does not necessarily apply for other types of DOD manipulation conflicts.

For domain-specific reasons or for reasons of efficiency of the resolution process, it may be the case that DOD modification focus determination selects a focus that includes only part of the known DOD assessment conflicts and DOD refinement conflicts. For example, there may be knowledge in DOD modification focus determination that assigns a priority to a conflict on the basis of the size of the ranges of possible values of domain object attributes involved in the conflict (e.g., a smaller size means a higher priority). If the selected focus does not include the DOD assessment conflict or the DOD refinement conflict that has just been detected, then the resolution of this conflict is effectively postponed. This corresponds to design practice, where it is not unusual that the designer focuses on a few conflicts right away and leaves the rest for later in the design process.

DOD manipulation conflict resolution. The following enumeration shows how each type of DOD manipulation conflict is resolved and by which design sub-process.

- *DOD assessment conflicts.* DOD modification determination resolves a DOD assessment conflict by selecting an alteration to the current design object description that deletes and/or changes domain object information involved in the conflict. (During this process, it may happen that a DOD modification conflict, a DOD modification method conflict, or a DOD alteration conflict occurs.)

- *DOD modification focus conflicts.* DOD modification focus determination resolves a DOD modification focus generation conflict by applying criteria for the selection of one of the DOD modification focus proposals. It may happen that a DOD modifica-

tion focus selection conflict occurs, which is resolved by selecting a DOD modification focus proposal with the largest sum of weights of supporting criteria.

- *DOD modification method conflicts.* DOD modification method determination resolves a DOD modification method generation conflict by applying criteria for the selection of one of the DOD modification method proposals. It may happen that a DOD modification method selection conflict occurs, which is resolved by selecting a DOD modification method proposal with the largest sum of weights of supporting criteria.

- *DOD alteration conflicts.* DOD modification method execution resolves a DOD alteration generation conflict by applying criteria for the selection of one of the DOD alteration proposals. It may happen that a DOD alteration selection conflict occurs, which is resolved by selecting a DOD alteration proposal with the largest sum of weights of supporting criteria.

- *DOD refinement conflicts.* DOD modification determination resolves a DOD refinement conflict by selecting an alteration to the current design object description that deletes and/or changes domain object information involved in the conflict. (During this process, it may happen that a DOD modification conflict, a DOD modification method conflict or a DOD alteration conflict occurs.)

In the following sub-sections, it is shown how the management of DOD manipulation conflicts can be specified in accordance with the model described above. For the purpose of illustration, examples from Section 14.2.2 are used.

14.4.1 Management of DOD assessment conflicts

Example 14.6 presents a satisfaction based DOD assessment conflict concerning the volume of a house. One requirement states that the volume of the house should be about 195 m^3, but the house has been designed to have a volume of 260 m^3. The management of this conflict is modelled as follows.

Detection of DOD assessment conflicts. In Example 14.6, the design requirements involved in the conflict are three requirements and two qualified requirements. These design requirements are modelled as follows.

Requirement R6a states that the design object must be a house, requirement R6b states that the house must have a volume of 195 m^3, and qualified requirement QR6a states that the requirements R6a and R6b must both be satisfied:

```
is-defined-as(R6a, has-value(house1, type, house))
is-defined-as(R6b, has-value(house1, volume, val(195, m^3)))
is-defined-as(QR6a, every, [R6a, R6b])
is-to-be-satisfied(QR6a)
```

Requirement R6c states that each floor of the house has a height of at least 2.60 meter, and qualified requirement QR6b states that the requirement R6c must be satisfied:

```
is-defined-as(R6c,
   for-all(N, l-implies(for-all(L, for-all(F, l-and(has-value(house1, floors, L),
        l-and(is-member-of(F, L), has-value(F, height, val(N, m)))))), ge(N, 2.60)))
is-defined-as(QR6b, every, [R6c])
is-to-be-satisfied(QR6b)
```

The DOD modification determination process within the DOD manipulation process is activated to reason with a current design object description that includes no domain object information. As a result, a design object description is produced with the domain object information that the design object is a house with two floors, each with a height of 2.6 meter and a volume of 50 m^3:

```
has-value(house1, type, house)
has-value(house1, floors, [floor1, floor2])
has-value(floor1, height, val(2.6, m))
has-value(floor1, area, val(50, m^2))
has-value(floor2, height, val(2.6, m))
has-value(floor2, area, val(50, m^2))
```

After deductive refinement of this design object description, resulting in the extra domain object information that the house has a total volume of 260 m^3, the current design object description is assessed. A satisfaction based DOD assessment conflict occurs, because requirement R6b is violated and, therefore, also qualified requirement QR6a.

After having assigned the highest priority to this single conflict, the next step for DOD manipulation is to resolve this conflict by modifying the current design object description.

Resolution of DOD assessment conflicts. Since the modification of design object descriptions is discussed extensively in the following sub-section, the DOD modification process to resolve the DOD assessment conflict in Example 14.6 is described briefly and informally. First of all, the DOD modification focus is set on the domain object information involved in the conflict, which concerns the number of floors, the height of a floor and the area of a floor.

Then the DOD modification method chosen is to replace (part of) the domain object information in focus by new domain object information. One of the two alteration proposals is to reduce the number of floors and to enlarge the area of the first floor to 75 m^2:

```
is-proposed-DOD-alteration(DOD-alteration6a)
includes-DOD-modification(DOD-alteration6a,
  deletion-of(domain-object-information(has-value(house1, floors, [floor1, floor2]), pos)))
includes-DOD-modification(DOD-alteration6a,
  deletion-of(domain-object-information(has-value(floor1, area, val(50, m^2)), pos)))
includes-DOD-modification(DOD-alteration6a,
  deletion-of(domain-object-information(has-value(floor2, height, val(2.6, m)), pos)))
includes-DOD-modification(DOD-alteration6a,
  deletion-of(domain-object-information(has-value(floor2, area, val(50, m^2)), pos)))
includes-DOD-modification(DOD-alteration6a,
  addition-of(domain-object-information(has-value(house1, floors, [floor1]), pos)))
includes-DOD-modification(DOD-alteration6a,
  addition-of(domain-object-information(has-value(floor1, area, val(75, m^2)), pos)))
```

A second alteration proposal is to reduce the area of the first floor to 45 m^2 and the area of the second floor to 30 m^2:

```
is-proposed-DOD-alteration(DOD-alteration6b)
includes-DOD-modification(DOD-alteration6b,
  deletion-of(domain-object-information(has-value(floor1, area, val(50, m^2)), pos)))
includes-DOD-modification(DOD-alteration6b,
  deletion-of(domain-object-information(has-value(floor2, area, val(50, m^2)), pos)))
includes-DOD-modification(DOD-alteration6b,
  addition-of(domain-object-information(has-value(floor1, area, val(45, m^2)), pos)))
includes-DOD-modification(DOD-alteration6b,
  addition-of(domain-object-information(has-value(floor2, area, val(30, m^2)), pos)))
```

To select one of these alteration proposals, the customer is asked to state his preferences. The customer prefers two floors to one. On the basis of this preference, the proposal DOD-alteration6b to reduce the area of the two floors is selected. Deductive refinement of the modified design object description results in a design object description, which includes the domain object information that the house has a total volume of 195 m^3.

14.4.2 Management of DOD modification conflicts

The management of DOD modification conflicts follows a distinct pattern. In the beginning of Section 14.4, it has been explained that the ways in which DOD modification focus conflicts, DOD modification method conflicts, and DOD alteration conflicts can be detected, prioritised, and resolved are comparable. Therefore, rather than elaborating the management of all different types of DOD modification conflicts, this sub-section presents specifications of knowledge for management of one type DOD modification conflict: DOD modification method conflicts.

Example 14.8 presents a DOD modification method generation conflict that occurs during re-design of the piping in a bathroom. One option is to modify the routings of the water supply system and the sewage discharge system simultaneously, and another option is to modify these routings sequentially.

During the selection of one of these options, a DOD modification method selection conflict occurs. According to the criterion of efficiency of the re-design process, the best choice for the modification method would be to revise the two routings sequentially, but according to the criterion of optimality of the solution (with respect to space occupancy), the best choice would be to re-design the routings simultaneously. The management of these two DOD modification method conflicts is modelled as follows.

Detection of DOD modification method conflicts. The DOD modification method generation conflict in Example 14.8 is detected as follows. First, an inventory is made of the possible DOD modification methods for re-designing the routing of the water supply system and the routing of the sewage discharge system. This task-specific process is modelled by the component DOD-modification-method-inventory, which is a sub-component of the component DOD-modification-method-determination.

In Example 14.8, one proposed method is to re-design the two routings sequentially, and another proposal is to re-design the routings simultaneously. This information is modelled as part of the output of the component DOD-modification-method-inventory:

```
is-proposed-DOD-modification-method(sequential-redesign)
is-proposed-DOD-modification-method(simultaneous-redesign)
```

The DOD modification method generation conflict is caused by the presence of multiple proposals for a new DOD modification method.

The DOD modification method selection conflict in Example 14.8 is detected as follows. Available domain-specific knowledge is that there are two applicable criteria: efficiency of the re-design process and optimality of the solution (with respect to space occupancy). A modification method meets the criterion of efficiency if (1) the modification focus includes one part, or if (2) the modification focus involves two or more parts, which must be re-designed sequentially. A modification method meets the criterion of optimality if (1) the modification focus includes one part, or if (2) the modification focus involves two or more

parts, which must be re-designed simultaneously. Available task-specific knowledge is that the best DOD modification method is one that meets all applicable criteria. Together, this application-specific knowledge is modelled by the following facts and rules within the knowledge base of the component DOD-modification-method-proposal-evaluation:

```
is-set-of-applicable-DOD-modification-method-selection-criteria([efficiency, optimality]);

has-size([], 0);

if has-size(L: DOD-part-list, N: number)
  and N1: number = N: number + 1
then has-size([P: DOD-part | L: DOD-part-list], N1: number);

meets-criterion(sequential-redesign, efficiency);

if is-selected-DOD-modification-focus(parts(L: DOD-part-list))
  and has-size(L: DOD-part-list, 1)
then meets-criterion(simultaneous-redesign, efficiency);

if is-selected-DOD-modification-focus(parts(L: DOD-part-list))
  and has-size(L: DOD-part-list, N: number)
  and N: number > 2
then not meets-criterion(simultaneous-redesign, efficiency);

meets-criterion(simultaneous-redesign, optimality);

if is-selected-DOD-modification-focus(parts(L: DOD-part-list))
  and has-size(L: DOD-part-list, 1)
then meets-criterion(sequential-redesign, optimality);

if is-selected-DOD-modification-focus(parts(L: DOD-part-list))
  and has-size(L: DOD-part-list, N: number)
  and N: number > 2
then not meets-criterion(sequential-redesign, optimality);

meets-all-criteria-from(M: DOD-modification-method, [ ]);

if meets-all-criteria-from(M: DOD-modification-method, L: criterion-list)
  and meets-criterion(M: DOD-modification-method, C: criterion)
then meets-all-criteria-from(M: DOD-modification-method, [C: criterion | L: criterion-list]);
```

```
if is-set-of-applicable-DOD-modification-method-selection-criteria(L: criterion-list)
  and meets-all-criteria-from(M: DOD-modification-method, L: criterion-list)
then is-best-DOD-modification-method(M: DOD-modification-method);
```

If, in Example 14.8, the component DOD-modification-method-proposal-evaluation is activated to reason with the information that the selected modification focus is the set of parts of the design object description that concern the routings of the water supply system and the sewage discharge system:

```
is-selected-DOD-modification-focus(parts([water-supply-system1, sewage-discharge-system1]))
```

and if the decision targets of this component are to confirm that the two given DOD modification method proposals are the best:

```
target(decision, is-best-DOD-modification-method(sequential-redesign), confirm)
target(decision, is-best-DOD-modification-method(simultaneous-redesign), confirm)
```

then its output includes the information that the criteria of efficiency and optimality apply, and that the two proposals to re-design routings each meet a different criterion:

```
is-set-of-applicable-DOD-modification-method-selection-criteria([efficiency, optimality])
meets-criterion(sequential-redesign, efficiency)
not meets-criterion(sequential-redesign, optimality)
not meets-criterion(simultaneous-redesign, efficiency)
meets-criterion(simultaneous-redesign, optimality)
```

The DOD modification method selection conflict is caused by the absence of a best proposal, since neither one of the DOD modification method proposals meets both applicable criteria. The conflict is modelled by the task control information that the component DOD-modification-method-proposal-evaluation has failed to confirm any of its decision targets.

Resolution of DOD modification method conflicts. The DOD modification method conflicts in Example 14.8 are resolved in three steps: weighing of DOD modification method proposals on the basis of applicable criteria, ranking of DOD modification method proposals on the basis of their weights, and selection of a best DOD modification method proposal.

Firstly, the applicable criteria are assigned weights, reflecting their importance in the selection of a DOD modification method proposal, given the current stage of the design process. Available domain-specific knowledge is that three design process stages exist: early, middle, and late. Efficiency of the re-design process is more preferred in a later stage of the design process, as it is more important to finish with some solution than to have an optimal

solution. Optimality of the re-design solution is preferred in an earlier stage, as it is probably more difficult to arrive at an optimal solution in a later stage, when time may be pressing.

Available task-specific knowledge is that the weight of a proposed DOD modification method equals the sum of weights of the criteria that it meets. Together, this application-specific knowledge is modelled by the following facts and rules within the knowledge base of the component DOD-modification-method-proposal-weighing, which is a sub-component of the component DOD-modification-method-determination:

```
;; Weights are assigned to criteria on the basis of the stage of the design process.

has-weight-in-design-stage(optimality, early-stage, 1.0);
has-weight-in-design-stage(optimality, middle-stage, 0.5);
has-weight-in-design-stage(optimality, late-stage, 0.0);
has-weight-in-design-stage(efficiency, early-stage, 0.0);
has-weight-in-design-stage(efficiency, middle-stage, 0.5);
has-weight-in-design-stage(efficiency, late-stage, 1.0);

is-member-of(C: criterion, [C: criterion | L: criterion-list]);

if is-member-of(C1: criterion, L: criterion-list)
then is-member-of(C1: criterion, [C2: criterion | L: criterion-list]);

if is-set-of-applicable-DOD-modification-method-selection-criteria(L: criterion-list)
  and is-member-of(C: criterion, L: criterion-list)
  and is-current-design-stage(DPS: design-process-stage)
  and has-weight-in-stage(C: criterion, DPS: design-process-stage, W: real)
then has-weight(C: criterion, W: real);

;; Weights are assigned to proposed methods on the basis of the criteria that they meet.

has-summed-weight(M: DOD-modification-method, [ ], 0.0);

if has-summed-weight(M: DOD-modification-method, L: criterion-list, OldW: real)
  and meets-criterion(M: DOD-modification-method, C: criterion)
  and has-weight(C: criterion, CriterionW: real)
  and W: real = OldW: real + CriterionW: real
then has-summed-weight(M: DOD-modification-method, [C: criterion | L: criterion-list], W: real);

if has-summed-weight(M: DOD-modification-method, L: criterion-list, W: real)
  and not meets-criterion(M: DOD-modification-method, C: criterion)
then has-summed-weight(M: DOD-modification-method, [C: criterion | L: criterion-list], W: real);
```

```
if is-proposed-DOD-modification-method(M: DOD-modification-method)
  and is-set-of-applicable-DOD-modification-method-selection-criteria(L: criterion-list)
  and has-summed-weight(M: DOD-modification-method, L: criterion-list, W: real)
then has-weight(M: DOD-modification-method, W: real);
```

If, in Example 14.6, the component DOD-modification-method-proposal-weighing is activated to reason with the following information: (1) the design process is in a middle stage, (2) there is one proposal to re-design the piping sequentially and one proposal to re-design the piping simultaneously, (3) the criteria of efficiency and optimality apply, and (4) the two proposals each meet different criteria:

```
is-current-design-stage(middle-stage)

is-proposed-DOD-modification-method(sequential-redesign)
is-proposed-DOD-modification-method(simultaneous-redesign)

is-set-of-applicable-DOD-modification-method-selection-criteria([efficiency, optimality])

meets-criterion(sequential-redesign, efficiency)
not meets-criterion(sequential-redesign, optimality)
not meets-criterion(simultaneous-redesign, efficiency)
meets-criterion(simultaneous-redesign, optimality)
```

then its output includes the information that both proposals have a weight of 0.5:

```
has-weight(sequential-redesign, 0.5)
has-weight(simultaneous-redesign, 0.5)
```

Secondly, the DOD modification method proposals are ranked on the basis of their weights, in order to determine the best DOD modification method: a heavier weight results in a higher rank, equal weights in the same rank. This task-specific knowledge is modelled by facts and rules within the knowledge base of the component DOD-modification-method-proposal-ranking, of which the details are omitted here (because of the large size of the specification). In Example 14.6, both proposals are the best DOD modification methods.

Thirdly, a proposed DOD modification method is selected that has been determined to be the best. This task-specific knowledge is modelled by a rule within the knowledge base of the component DOD-modification-method-proposal-selection, which is a sub-component of the component DOD-modification-method-determination:

```
if is-best-DOD-modification-method(M: DOD-modification-method)
then is-selected-DOD-modification-method(M: DOD-modification-method);
```

The targets of this component are to confirm the selection of proposed DOD modification methods. It may happen that there is more than one best method, in which case one of them has to be selected at random. This knowledge is specified by the initial evaluation criterion any of the component DOD-modification-method-proposal-selection, stating that the component may terminate its activation when it has achieved any of its targets.

In Example 14.6, the method to re-design the water supply system and the sewage discharge system simultaneously is selected at random as the new DOD modification method, which ends the resolution of the two DOD modification method conflicts.

14.4.3 Management of DOD refinement conflicts

Example 14.10 presents a deductive DOD refinement conflict concerning the discharge channel of a central heating furnace. According to the current design object description, the furnace is connected to an air discharge channel, but according to the domain knowledge, the channel to which the furnace is connected is a gas discharge channel. The management of this conflict is modelled as follows.

Detection of DOD refinement conflicts. In Example 14.10, the domain object information included in the design object description, that forms the basis of the deductive DOD refinement conflict, is that there is a furnace that is connected to an air discharge channel:

```
has-value(furnace1, type, furnace)
has-value(furnace1, discharge-connection, discharge-channel1)
has-value(discharge-channel1, type, air-discharge-channel)
```

The DOD manipulation process deductively refines the design object description with this domain object information Available domain-specific knowledge is that a furnace is always connected to a gas discharge channel. This domain-specific knowledge is modelled by the following rule within the knowledge base of the component deductive-DOD-refinement:

```
if has-value(O1: domain-object, type, furnace)
  and has-value(O1: domain-object, discharge-connection, O2: domain-object)
then has-value(O2: domain-object, type, gas-discharge-channel);
```

If, in Example 14.10, the component deductive-DOD-refinement is activated to reason with the information that the furnace is connected to an air discharge channel, then its output in-

cludes the domain object information that the channel to which the furnace is connected is an air discharge channel as well as a gas discharge channel:

```
has-value(furnace1, type, furnace)
has-value(furnace1, discharge-connection, discharge-channel1)
has-value(discharge-channel1, type, air-discharge-channel)
has-value(discharge-channel1, type, gas-discharge-channel)
```

The deductive DOD refinement conflict (detected by the DOD assessment process) is caused by the presence of two values for the type of the single discharge channel, and one of these values is the result of deductive refinement.

After having assigned the highest priority to this single DOD refinement conflict, the next step for DOD manipulation is to resolve the DOD refinement conflict by modifying the given design object description.

Resolution of DOD refinement conflicts. The conflict in Example 14.10 is resolved as follows. (As the modification of design object descriptions has been discussed extensively in the previous sub-section, the DOD modification process for Example 14.10 is described briefly and informally.) First of all, domain-specific knowledge is used to determine a suitable DOD modification focus on the given design object description. The obvious DOD modification focus is a set with the domain object information about the discharge channel connected to the furnace.

Knowing that activation of the deductive DOD refinement process has resulted in the new information that the discharge channel is a gas discharge channel, a simple method to resolve the conflict is to remove the domain object information that the specific discharge channel is an air discharge channel. Hence, the selected alteration is the following:

```
is-selected-DOD-alteration(DOD-alteration10)
includes-DOD-modification(DOD-alteration10, deletion-of(
   domain-object-information(has-value(discharge-channel1, type, air-discharge-channel), pos)))
```

After modification of the current design object description, activation of DOD assessment results in the conclusion that there are no more DOD manipulation conflicts.

14.5 Management of Design Process Co-ordination Conflicts

The generic model of design, GDM, has been used to model and specify the management of the different types of design process co-ordination conflicts described in Section 14.2.3. Design process co-ordination conflicts are managed entirely by the design process co-ordination process of a design process.

- **Design process co-ordination conflict detection.** If design process co-ordination concludes that the design process has violated one of the given design process objectives, then a design process objective satisfaction conflict occurs. If design process co-ordination generates multiple proposals for an overall design strategy, then a design strategy generation conflict occurs. Subsequently, if design process co-ordination applies criteria to select an overall design strategy from the proposals, and the number of proposals that meet all of these criteria does not equal one, then a design strategy selection conflict occurs.

- **Design process co-ordination conflict prioritisation.** There is no need to prioritise design process objective satisfaction conflicts (e.g., a deadline that has not been met, or specific design resources that have been exhausted), for they cannot be resolved. Design process objectives refer to the design process itself, and the history of a design process cannot be changed. Design strategy conflicts are resolved on a *Last In First Out* basis, which means that attempts to resolve a conflict start when the conflict is detected.

- **Design process co-ordination conflict resolution.** Design process co-ordination resolves a design strategy generation conflict by applying criteria for the selection of one of the overall design strategy proposals. During this process, it may happen that a design strategy selection conflict occurs, which design process co-ordination resolves by selecting an overall design strategy proposal with the largest sum of weights of supporting criteria.

In the remainder of this section, it is shown how the management of design process co-ordination conflicts can be specified in accordance with the model described above. For the purpose of illustration, examples from Section 14.2.3 are used.

Example 14.11 presents a design strategy generation conflict concerning an architect having to decide in an early stage of the design process which strategy to follow. One option is to first negotiate the conflicting customer requirements with the customer and then make a satisfactory and complete design, and another option is to first make a partial design that satisfies the non-conflicting requirements, then negotiate the conflicting customer requirements with the customer, and finally finish the design.

During the selection of one of these options, a design strategy selection conflict occurs. Both the criterion of early customer involvement and the criterion of late customer involvement apply. According to the first criterion, the best choice would be to negotiate the conflicting customer requirements with the customer first. According to the second criterion, the best choice would be to make a partial design first and then negotiate the conflicting requirements. The management of these two design strategy conflicts is modelled as follows.

Detection of design strategy conflicts. The design strategy generation conflict in Example 14.11 is detected as follows. First, the possible overall design strategies are generated. This process is modelled by means of the component overall-design-strategy-generation, which is a sub-component of the component DPC.

In Example 14.11, there is a proposal to first negotiate the conflicting requirements with the customer, and a proposal to first make a partial design for the non-conflicting requirements and then negotiate the conflicting requirements with the customer. This information is modelled as part of the output the component overall-design-strategy-generation:

```
is-proposed-overall-design-strategy(ODS11a)
is-proposed-overall-design-strategy(ODS11b)
is-defined-as(ODS11a,
  do-in-sequence(negotiate-conflicting-customer-requirements, finish-design))
is-defined-as(ODS11b,
  do-in-sequence(make-partial-design,
    do-in-sequence(negotiate-conflicting-customer-requirements, finish-design)))
```

The design strategy generation conflict is caused by the presence of multiple proposals for a new overall design strategy.

The design strategy selection conflict in Example 14.11 is detected as follows. Available domain-specific knowledge is that there are two applicable criteria: early involvement of the customer in the design process, and late involvement of the customer in the design process. An overall design strategy meets early customer involvement if the customer's conflicting requirements (if any) are negotiated in an early stage, and it meets late customer involvement if the customer's conflicting requirements (if any) are negotiated in a later stage.

Available task-specific knowledge is that the best overall design strategy is one that meets all applicable criteria. Together, this application-specific knowledge is modelled by the following rules within the knowledge base of the component overall-design-strategy-proposal-evaluation, which is a sub-component of the component DPC:

```
is-set-of-applicable-overall-design-strategy-selection-criteria(
  [early-customer-involvement, late-customer-involvement]);

is-part-of(P: control-process-plan, P: control-process-plan);

if is-part-of(P1: control-process-plan, P2: control-process-plan)
then is-part-of(P1: control-process-plan,
  if-then(E: design-process-results-info-expression, P2: control-process-plan));
```

```
if is-part-of(P1: control-process-plan, P2: control-process-plan)
then is-part-of(P1: control-process-plan,
  if-then-else(E: design-process-results-info-expression,
    P2: control-process-plan, P3: control-process-plan));

if is-part-of(P1: control-process-plan, P3: control-process-plan)
then is-part-of(P1: control-process-plan,
  if-then-else(E: design-process-results-info-expression,
  P2: control-process-plan, P3: control-process-plan));

if is-part-of(P1: control-process-plan, P2: control-process-plan)
then is-part-of(P1: control-process-plan,
  while-do(E: design-process-results-info-expression, P2: control-process-plan));

if is-part-of(P1: control-process-plan, P2: control-process-plan)
then is-part-of(P1: control-process-plan,
  repeat-until(P2: control-process-plan, E: design-process-results-info-expression));

if is-part-of(P1: control-process-plan, P2: control-process-plan)
then is-part-of(P1: control-process-plan,
  do-in-sequence(P2: control-process-plan, P3: control-process-plan));

if is-part-of(P1: control-process-plan, P3: control-process-plan)
then is-part-of(P1: control-process-plan,
  do-in-sequence(P2: control-process-plan, P3: control-process-plan));

if is-part-of(P1: control-process-plan, P2: control-process-plan)
then is-part-of(P1: control-process-plan,
  do-in-parallel(P2: control-process-plan, P3: control-process-plan));

if is-part-of(P1: control-process-plan, P3: control-process-plan)
then is-part-of(P1: control-process-plan,
  do-in-parallel(P2: control-process-plan, P3: control-process-plan));

if is-current-overall-design-process-state(S: design-process-state)
  and includes-design-process-results-information(
    S: design-process-state, is-current-design-stage(early-stage), pos)
  and is-defined-as(OverallDSN: design-strategy-name,
    do-in-sequence(negotiate-conflicting-customer-requirements, P: control-process-plan))
then meets-criterion(OverallDSN: design-strategy-name, early-customer-involvement);
```

```
if is-current-overall-design-process-state(S: design-process-state)
  and includes-design-process-results-information(
    S: design-process-state, is-current-design-stage(early-stage), pos)
  and is-defined-as(OverallDSN: design-strategy-name,
    do-in-sequence(P1: control-process-plan, P2: control-process-plan))
  and not P1: control-process-plan = negotiate-conflicting-customer-requirements
  and is-part-of(negotiate-conflicting-customer-requirements, P2: control-process-plan)
then not meets-criterion(OverallDSN: design-strategy-name, early-customer-involvement);

if is-current-overall-design-process-state(S: design-process-state)
  and includes-design-process-results-information(
    S: design-process-state, is-current-design-stage(early-stage), neg)
  and is-defined-as(OverallDSN: design-strategy-name,
    do-in-sequence(P1: control-process-plan, P2: control-process-plan))
  and not P1: control-process-plan = negotiate-conflicting-customer-requirements
  and is-part-of(negotiate-conflicting-customer-requirements, P2: control-process-plan)
then meets-criterion(OverallDSN: design-strategy-name, late-customer-involvement);

if is-current-overall-design-process-state(S: design-process-state)
  and includes-design-process-results-information(
    S: design-process-state, is-current-design-stage(early-stage), neg)
  and is-defined-as(OverallDSN: design-strategy-name,
    do-in-sequence(negotiate-conflicting-customer-requirements, P: control-process-plan))
then not meets-criterion(OverallDSN: design-strategy-name, late-customer-involvement);

meets-all-criteria-from(OverallDSN: design-strategy-name, [ ]);

if meets-all-criteria-from(OverallDSN: design-strategy-name, L: criterion-list)
  and meets-criterion(OverallDSN: design-strategy-name, C: criterion)
then meets-all-criteria-from(OverallDSN: design-strategy-name, [C: criterion | L: criterion-list]);

if is-set-of-applicable-overall-design-strategy-selection-criteria(L: criterion-list)
  and meets-all-criteria-from(OverallDSN: design-strategy-name, L: criterion-list)
then is-best-overall-design-strategy(OverallDSN: design-strategy-name);
```

If, in Example 14.11, the component overall-design-strategy-proposal-evaluation is activated to reason with the information that the current design process state (named State11) includes the design process information that the design process is in an early stage:

```
is-current-overall-design-process-state(State11)
includes-design-process-results-information(State11, is-current-design-stage(early-stage), pos)
```

and if the decision targets of this component are to confirm that the two given overall design strategy proposals are the best:

```
target(decision, is-best-overall-design-strategy(ODS11a), confirm)
target(decision, is-best-overall-design-strategy(ODS11b),  confirm)
```

then its output includes the information that the criteria of early customer involvement and late customer involvement apply, and that the two proposals each meet a different criterion:

```
is-set-of-applicable-overall-design-strategy-selection-criteria(
  [early-customer-involvement, late-customer-involvement])
meets-criterion(ODS11a, early-customer-involvement)
not meets-criterion(ODS11a, late-customer-involvement)
not meets-criterion(ODS11b, early-customer-involvement)
meets-criterion(ODS11b, late-customer-involvement)
```

The design strategy selection conflict is caused by the absence of a best proposal, since neither one of the overall design strategy proposals meets all applicable criteria. The conflict is modelled by the task control information that the component overall-design-strategy-proposal-evaluation has failed to confirm its decision targets.

Resolution of design strategy conflicts. The design strategy conflicts in Example 14.11 are resolved in three steps: weighing of overall design strategy proposals on the basis of applicable criteria, ranking of overall design strategy proposals on the basis of their weights, and selection of a best overall design strategy proposal.

Firstly, the applicable criteria are assigned weights, reflecting their importance in the selection of a proposal. Available domain-specific knowledge is that early customer involvement is preferred over late customer involvement, as it may be time-consuming to make modifications first while chances are that the customer turns it down (perhaps even unseen).

Available task-specific knowledge is that the weight of a proposed overall design strategy equals the sum of weights of the criteria that it meets. Together, this application-specific knowledge is modelled by the following facts and rules within the knowledge base of the component overall-design-strategy-proposal-weighing, which is a sub-component of the component DPC:

```
has-weight(early-customer-involvement, 1.0);
has-weight(late-customer-involvement, 0.0);

has-summed-weight(DSN: design-strategy-name, [ ], 0.0);
```

```
if has-summed-weight(DSN: design-strategy-name, L: criterion-list, OldW: real)
  and meets-criterion(DSN: design-strategy-name, C: criterion)
  and has-weight(C: criterion, CriterionW: real)
  and W: real = OldW: real + CriterionW: real
then has-summed-weight(DSN: design-strategy-name, [C: criterion | L: criterion-list], W: real);

if has-summed-weight(DSN: design-strategy-name, L: criterion-list, W: real)
  and not meets-criterion(DSN: design-strategy-name, C: criterion)
then has-summed-weight(DSN: design-strategy-name, [C: criterion | L: criterion-list], W: real);

if is-proposed-overall-design-strategy(DSN: design-strategy-name)
  and is-set-of-applicable-overall-design-strategy-selection-criteria(L: criterion-list)
  and has-summed-weight(DSN: design-strategy-name, L: criterion-list, W: real)
then has-weight(DSN: design-strategy-name, W: real);
```

If, in Example 14.11, the component overall-design-strategy-proposal-weighing is activated to reason with the following information: (1) there is one proposal to negotiate the conflicting customer requirements first and one proposal to postpone the negotiation with the customer about the conflicting customer requirements, (2) the criteria of early customer involvement and late customer involvement apply, and (3) the proposal to negotiate the conflicting customer requirements first meets early customer involvement but not late customer involvement, and the proposal to postpone the negotiation with the customer about the conflicting customer requirements meets late customer involvement but not early customer involvement:

```
is-proposed-overall-design-strategy(ODS11a)
is-proposed-overall-design-strategy(ODS11b)

is-defined-as(ODS11a,
  do-in-sequence(negotiate-conflicting-customer-requirements, finish-design))

is-defined-as(ODS11b,
  do-in-sequence(make-partial-design,
    do-in-sequence(negotiate-conflicting-customer-requirements, finish-design)))

is-set-of-applicable-overall-design-strategy-selection-criteria(
  [early-customer-involvement, late-customer-involvement])

meets-criterion(ODS11a, early-customer-involvement)
not meets-criterion(ODS11a, late-customer-involvement)
not meets-criterion(ODS11b, early-customer-involvement)
meets-criterion(ODS11b, late-customer-involvement)
```

then its output includes the information that the proposal to negotiate the conflicting customer requirements first has a weight of 1.0 and the proposal to postpone the negotiation with the customer about the conflicting customer requirements has a weight of 0.0:

```
has-weight(ODS11a, 1.0)
has-weight(ODS11b, 0.0)
```

Secondly, the overall design strategy proposals are ranked on the basis of their weights, in order to determine the best overall design strategy: a heavier weight results in a higher rank, and equal weights in the same rank. This task-specific knowledge is modelled by facts and rules within the knowledge base of the component overall-design-strategy-proposal-ranking, of which the details are omitted here (because of the large size of the specification). In Example 14.11, the best overall design strategy is to negotiate the conflicting customer requirements first.

Thirdly, a proposed overall design strategy determined to be the best is selected. This task-specific knowledge is modelled by a rule within the knowledge base of the component overall-design-strategy-proposal-selection, which is a sub-component of the component DPC:

```
if is-best-overall-design-strategy(DSN: design-strategy-name)
then is-selected-overall-design-strategy(DSN: design-strategy-name);
```

The targets of this component are to confirm the selection of proposed overall design strategies. It may happen that there is more than one best strategy, in which case one of them has to be selected at random. This knowledge is specified by the initial evaluation criterion any of the component overall-design-strategy-proposal-selection, stating that the component may terminate its activation when it has achieved any of its targets.

In Example 14.11, the strategy to negotiate the conflicting customer requirements first is selected as the new overall design strategy, which ends the resolution of the two design strategy conflicts.

14.6 Discussion

Conflicts are inherent to design. In general, a design conflict is an inconsistency between specific elements of design information. The view underlying conflict management in design in this chapter is that design is a process of detecting, prioritising and resolving design conflicts. That is, a design process proceeds by a series of attempts to generate new requirement qualification sets and design object descriptions, as a result of which conflicts occur. Since not all conflicts need to be resolved immediately or cannot be resolved at the same time, the detected conflicts are prioritised, and conflicts are resolved in an order based on their priority.

The complex reasoning processes involved in conflict management in design are highly dynamic and non-monotonic. Design conflicts often evolve in the sense that the resolution of one conflict results in the creation of another, that, in turn, may (have to) be resolved immediately or may be ignored for a while. However, for a design process to end successfully, there may be no design conflicts left, regardless of which overall design strategy is effective.

AI in Design researchers have addressed the problem of providing computational support for conflict management in design support systems, both for single-agent and multi-agent design processes. In this chapter, a few notable approaches have been described:

- conflict management by design and resolution ([Oh and Sharpe, 1995]),
- conflict management by integrated exception handling ([Klein, 1995]),
- conflict management by tracking Pareto optimality ([Petrie, Webster and Cutkosky, 1995]),
- conflict management by negotiation between single-function agents ([Dunskus, Grecu, Brown and Berker, 1995]),
- conflict management by assumption-based constraint satisfaction ([Haroud, Boulanger, Gelle and Smith, 1995]).

Most of these approaches recognise the importance of providing services for capturing design decisions and their interdependencies for conflict resolution purposes. What is missing is an elaborate typology of design conflicts that a (single) design agent might encounter, as well as a thorough treatment of the complex reasoning involved in conflict management.

The three components of the generic task model GDM provide a basis for the distinction of three main types of conflict encountered in a design process: RQS manipulation conflicts, DOD manipulation conflicts and design process co-ordination conflicts. As GDM is a model of a single agent's design process, it does not distinguish co-ordination conflicts between multiple design agents, which are the fourth main type of conflict in a design process.

This chapter has demonstrated how GDM can be used as a basis for modelling the design sub-processes involved as well as for specifying the knowledge for the detection, prioritisation, and resolution of single-agent design conflicts. Examples have been given for the management of RQS manipulation conflicts, DOD manipulation conflicts, and design process co-ordination conflicts.

Chapter 15

Contributions to AI in Design Research

This chapter reviews the results of this thesis' research on the anatomy of design processes. It describes the contributions of this thesis to the research in the field of Artificial Intelligence in Design with respect to design theories and models, design methods and design support systems, as well as the limitations.

This thesis has taken the stance that, despite the ill structured nature of design problems, a design process need not necessarily be ill structured. The central hypothesis of this thesis, as formulated in Chapter 1, is that a design process has a well defined anatomy: a generic structure that is independent of its application domain, the design problem at hand, and the repertory of available design methods and techniques. Our claim has been that investigating this anatomy contributes to a better understanding of design processes, which furthers the development of a theory of design and the development of useful design support systems.

This chapter reviews the extent to which this thesis has provided evidence that supports the central hypothesis. It describes the contributions made by this thesis to AI in Design research on design theories, models, methods and support systems, using Chapter 2 as a point of reference. It further describes the limitations of this thesis' research on the anatomy of design processes, which lays the ground for the directions for further AI in Design research presented in Chapter 16.

This chapter is organised as follows. Section 15.1 describes how this thesis contributes to the development of design theories and models. Section 15.2 reviews this thesis' contributions to the development of design methods, and Section 15.3 does the same for the development of design support systems.

15.1 Design Theories and Models

This thesis provides an analysis of design processes on the knowledge level, using logic as a tool. As pointed out in Chapter 1, a theory of design cannot be proven to be correct and complete in a mathematical sense; rather, it is at best consistent with the known results from empirical studies. Therefore, our approach has been to iteratively develop a knowledge-level theory of design processes and a corresponding generic model (as a computational counterpart of the theory) and subsequently test this generic design model in practice.

Our logical theory of design, and in particular our generic design model GDM, have proven to stand the test in many practical situations. This section extensively discusses their value.

15.1.1 Logical theory of design

The main advantage of formulating a theory on the knowledge level is that it can be interpreted independent of procedural and representational issues. Part II of this thesis presented a logical theory of design, which describes the anatomy of design as a process on the knowledge level, for which purpose (first-order predicate) logic has been used as a vehicle.

Chapter 3 described the static aspects of design, which concern the possible descriptions involved in a design process, such as sets of design requirements, or design object descriptions. Chapter 4 described the dynamic aspects of design, which concern the possible states and the sequences of state transitions within a design process, such as modification of a set of design requirements, or deductive refinement of a design object description.

As described in Chapter 2, a theory or model of design should define knowledge (i.e., definitions and axioms) about a number of concepts. Of these concepts, our logical theory of design defines the following:

- object, or entity,
- object type, or object class,
- property, or attribute,
- behaviour,
- composition or structure of an object,
- semantic relationship between objects,
- domain knowledge,
- context, environment, or field,
- function (of an object),
- need, or desire,
- requirement, design criterion, or (functional) design specification,
- design problem, requirements description, or design program,
- design problem space,
- design solution, design/artefact description, or product design specification,

- design solution space,
- design activity, design event, design (sub-)task, design operation, or design function,
- design process, or design project,
- design state, or design stage,
- design step, design move, design change or design revision,
- design (control) strategy, or design scenario,
- design decision,
- design conflict,
- design trace,
- design pattern,
- design rationale,
- design history.

Concepts that are not defined by our logical theory are:

- creativity in design,
- situatedness in design,
- learning in design,
- distributed design.

Therefore, it can be concluded that our logical theory of design covers most of the concepts that design researchers agree to be relevant to design processes, except for concepts such as creativity in design, situatedness in design, learning in design, and distributed design.

15.1.2 Generic design model

To make the step to practical applications, a generic model of a process is very useful. It is in general not as powerful as a theory, for it may not capture all aspects of a given type of process. But the price of a restricted applicability is usually more than compensated by something that the theory may lack: the possibility of being implemented (fairly straightforwardly) on a computer. Such an implementation offers possibilities for analysis, simulation, automation and support of human users.

Hence, Part III of this thesis presented a generic design model, called GDM, which is a computational counterpart of the logical theory presented in Part II. GDM is a blueprint of the generic features of design processes: it models the significant types of information and knowledge that play a role within a design process, irrespective of the specific design method and application domain. By using GDM to model a specific design process, a developer can focus directly on the design method and application domain. The developer may opt to specialise GDM by including method-specific knowledge and to instantiate GDM by including domain-specific knowledge. Our claim is that this approach is a powerful means to analyse complex design processes and to develop design support systems in a well-founded manner.

GDM models a design process as a whole and its three main sub-processes: requirement qualification set manipulation, design object description manipulation and design process coordination. A summary of these four processes and a discussion of the qualities and limitations of GDM with respect to these four processes are provided below.

Design. Chapter 6 described the two upper process abstraction levels of GDM. This model defines the design process, which is composed of three sub-processes: a design process coordination process to generate a successful overall design strategy, a requirement qualification set manipulation process to generate a well-structured set of design requirements, and a design object description manipulation process to generate a satisfactory design object description.

In a design process, four levels of reflection are distinguished. From the bottom to the top, these levels are:

- the object level, with information and knowledge about design objects,
- the first meta-level, with information and knowledge about design requirements and about design object descriptions,
- the second meta-level, with information and knowledge about requirement qualification sets and about modifications and assessments of requirement qualifications sets and design object descriptions, respectively, and
- the third meta-level, with information and knowledge about design process objectives, overall design strategies, and evaluations of the RQS manipulation process, the DOD manipulation process, and the design process as a whole.

The distinction of four reflection levels within a design process is a unique feature of GDM: there is no other model that covers more than two reasoning levels for knowledge processing activities within a design process. Especially the third meta-level is a novel concept, where design process objectives are modelled (e.g., requirements on the duration of the design process) as well as the knowledge to determine an overall design strategy to be followed (which may depend on the given design process objectives).

Requirement qualification set manipulation. Chapter 7 described GDM in more detail for the sub-process of manipulating requirement qualification sets within a design process. This sub-process (abbreviated RQS manipulation) is responsible for a well-structured set of design requirements that properly describe the function of the design object, its behaviour or shape.

An RQS manipulation process is composed of an RQS manipulation history maintenance process to keep track of historical RQS manipulation process information, an RQS modification process to generate a well-structured set of design requirements, a current RQS maintenance process to keep track of the current requirement qualification set, and a deductive RQS refinement process to deductively refine the current requirement qualification set by means of a design requirements theory.

In an RQS manipulation process, three levels of reflection are distinguished. From the bottom to the top, these levels are:

- the first meta-level, with information and knowledge about design requirements,
- the second meta-level, with information and knowledge about requirement qualification sets, about assessments of requirement qualifications sets and about the modification of requirement qualification sets, and
- the third meta-level, with information and knowledge about overall design strategies, RQS manipulation traces and RQS manipulation process evaluations.

At a first glance, it may be confusing that in GDM, the concept of requirement qualification set is used to denote a set of needs, desires, requirements, constraints and/or preferences. Although these different concepts appear frequently in design research literature (see Chapter 2), with different meanings, they are usually used only in an informal sense, lacking a formal definition. Regardless of the terminology used, all of these concepts are used to describe, in more or less detail, directly or indirectly, what the design artefact to be made should do. Therefore, it is useful to have one concept subsuming all of the other more specific concepts.

The distinction of a deductive RQS refinement sub-process within an RQS manipulation process is a unique feature of GDM. This sub-process accounts for the application of regulatory design knowledge such as fire safety regulations. For example, as soon as it becomes clear in a design process that the design artefact should be a building of a specific type, the deductive RQS refinement sub-process within that design process may automatically add the appropriate fire safety regulations for that type of building to the current requirement qualification set. That is, the presence of one design requirement automatically entails the introduction of others—GDM is the only model that addresses this type of process.

Design object description manipulation. Chapter 8 described GDM in more detail for the sub-process of manipulating design object descriptions within a design process. This sub-process (abbreviated DOD manipulation) is responsible for generating a consistent description of the structure, form and behaviour of the design object to be created. The description should include sufficient information for actually creating the design object.

A DOD manipulation process is composed of a DOD manipulation history maintenance process to keep track of historical DOD manipulation process information, a DOD modification process to generate a satisfactory design object description, a current DOD maintenance process to keep track of the current design object description, and a deductive DOD refinement process to deductively refine the current design object description by means of a domain objects theory.

In a DOD manipulation process, four levels of reflection are distinguished. From the bottom to the top, these levels are:

- the object level (hidden within DOD manipulation), with information and knowledge about domain objects,
- the first meta-level, with information and knowledge about design object descriptions,
- the second meta-level, with information and knowledge about requirement qualification sets, about assessments of design object descriptions and requirement qualification sets and about the modification of design object descriptions, and
- the third meta-level, with information and knowledge about overall design strategies, DOD manipulation traces and DOD manipulation process evaluations.

Including assessments of both design object descriptions and requirement qualification sets in the output of a DOD modification process is a unique feature of GDM. During DOD manipulation, it might at some point be concluded that no design object description can be found or generated that will satisfy a certain requirement (because of a logical inconsistency in the requirement or due to practical unfeasibility). This conclusion does not concern design object descriptions, but a specific requirement of the given requirement qualification set; therefore, the conclusion is an example of an RQS assessment. GDM is the only model that includes the assessment of requirement qualification sets as an output of a DOD modification process.

Design process co-ordination. Chapter 6 described a design process co-ordination process as one that aims to control a design process in accordance with the given design process objectives and an overall design strategy. This strategy prescribes to a greater or lesser degree how the design process has to proceed. GDM is one of the few models that addresses this type of process, as a recognition of the fact that design strategies are key to the successful completion of design processes.

Besides the description in Chapter 6, Chapter 12 presented an example of how GDM (and particularly its component for design process co-ordination) can be used in modelling strategic interactions between a designer and a design support system, and the strategic knowledge involved in such interactions. However, GDM describes design process co-ordination in less detail than RQS manipulation and DOD manipulation.

15.2 Design Methods

Part III of this thesis describes a generic design model that can be used to analyse and model design processes. Chapter 9 presented specialisations of GDM that are useful in the analysis of many practical design processes; such specialisations can help to model design methods applied in practice. However, as a knowledge-level model, for a given application domain GDM does not address the issue of representing a design method in the best possible way, in terms of the design problem space, the design solution space and the design knowledge.

What would be helpful is a set of guidelines that describe which types of representations would best fit which (specialised) parts of GDM. Such guidelines could enhance the process of analysing a specific design method, by providing clues for the recognition of the process composition, knowledge composition and relation between process composition and knowledge composition, given the representations employed in the design method. The guidelines could also enhance the process of developing a design method, by providing clues for the generation of representations to be employed in the design method, given the process composition, knowledge composition and relation between process composition and knowledge composition in the design process model based on GDM.

15.3 Design Support Systems

Chapter 5 described DESIRE, a compositional multi-agent design method that supports structured analysis and design of autonomous interactive agents, both for individual, agent-specific tasks, and for communication between agents. DESIRE views the individual agents, their tasks and the overall system as compositional structures and supports the evolutionary development of multi-agent systems. Therefore, DESIRE is a useful method for developing design support systems, as the end user and the design support system can both be well viewed as a design agent, each having its own tasks, and with a user interface to facilitate communication between the two agents.

Developed by means of DESIRE, the generic design model GDM is a useful starting point for the development of design support systems. It offers a process composition and a knowledge composition, as well as relations between those two compositions, which play a role within any design process. GDM may be part of a shared model of design between the design support system and the end user. Such a shared model is likely to be a more detailed version of GDM, as its process composition and knowledge composition are bound to incorporate features of a specific application domain and a specific design method.

With GDM as a starting point for the development and DESIRE as a compositional multi-agent design method, it is possible to model part of the functionality of a design support system, specifically:

- consistency maintenance, through the manipulation history maintenance components within RQS manipulation and DOD manipulation,
- context management, through the modification components within RQS manipulation and DOD manipulation,
- inference, through the modification components and the deductive refinement components within RQS manipulation and DOD manipulation, and
- criticising, through the modification components within RQS manipulation and DOD manipulation.

However, GDM does not provide guidance for user interfacing and knowledge representation. What is missing is a design application development environment for a developer of a design support system. Such an environment should include:

- a facility for requirements engineering, to acquire and model the requirements of a design support system,
- a library with representations of design methods, to be able to build design support systems that are required to support specific design methods,
- a modelling and specification facility, to generate a model and a detailed specification of a design support system (for which purpose GDM and its specialisations may be used as a point of departure), and
- a programming facility, to translate the specification of a design support system into a computer programming language such as Java.

Chapter 16

Directions for Further AI in Design Research

This chapter presents directions for further research in the field of Artificial Intelligence in Design. These directions concern design theories and models, design methods and design support systems.

This thesis has shown that a design process has a well defined anatomy: a generic structure that is independent of its application domain, the design problem at hand and the repertory of available design methods and techniques. Investigating this anatomy contributes to a better understanding of design processes and therefore to the development of a theory of design as a process as well as the development of useful design support systems. This is not to say, however, that every possible subject of research on design processes has been (sufficiently) dealt with. Hence, this chapter describes a few subjects worthy of investigation in further AI in Design research, given the contributions and limitations of this thesis' research described in Chapter 15.

This chapter is organised as follows. Following the same division of research subjects as used in Chapters 2 and 15, Section 16.1 presents directions for further research on design theories and models, Section 16.2 for design methods and Section 16.3 for design support systems.

16.1 Design Theories and Models

This section provides suggestions for further research on extensions to (and improvements of) our logical theory of design and our generic design model, GDM.

16.1.1 Logical theory of design

As pointed out in Chapter 15, the logical theory presented in Part II does not define all concepts considered to be relevant to design processes. Concepts that have not been defined are creativity in design, learning in design, situatedness in design and distributed design. This sub-section describes some directions for further research on each of these concepts, with references to publications in the same area of research.

16.1.1.1 Creativity in design

Creativity has been, and still is, a subject of research in scientific disciplines such as Philosophy, Artificial Intelligence and AI in Design. Boden has inspired many design researchers to make (cognitive) models of design with her work on the nature of creativity and the cognitive mechanisms and structures that play a role in creativity [Boden, 1990]. Already many years earlier, Wallas presented a model of creativity, consisting of four stages [Wallas, 1926]:

1. *Preparation*: researching the situation, searching for information.
2. *Incubation*: sleeping on it, taking a walk, or just reflecting.
3. *Illumination*: verbalising the intuitive understanding.
4. *Verification*: confirming the validity of the discovery.

As of yet, it is a challenge for research how to include these stages in a (knowledge-level) theory of design. Especially incubation and illumination can be expected to prove difficult to capture, as it is still unclear what information flows between these stages (if it is information at all) and which (cognitive) sub-processes are involved. As Wallas' model underlies many theories and models of creativity in design, it is a worthy point of departure for further design research, together with the results of research on creativity such as presented at AI in Design conferences. Specifically, it will be interesting to investigate how our logical theory of design can be extended to define creativity in design.

16.1.1.2 Learning in design

According to some design researchers, especially those from the machine learning in design community, design is inextricably linked with learning. According to Sim and Duffy, learning may take place after completion of a design activity (i.e., retrospective learning), in parallel with a design activity in progress (i.e., in-situ learning) and before the start of a design activity (i.e., provisional learning) [Sim and Duffy, 2000]. The learning goal associated with a design activity is to increase the efficacy or efficiency of the design activity.

For example, Gero proposes a paradigm for the formation of design concepts on the basis of the emergence of patterns in the representation of designs [Gero, 1998]. Emergence here means that design patterns are observed that have not been consciously constructed. By identifying and storing regularities in design states, new design concepts are learned that can be used in later design tasks, hopefully resulting in more efficient design processes.

It will be interesting to investigate how our logical theory of design can be extended to define learning in design. Specifically, it will require further exploration of the role of design history, considering it to be an active memory with learning capabilities rather than a passive memory with purely storage and retrieval duties. In the research, attention should be paid to retrospective learning, in-situ learning and provisional learning.

16.1.1.3 Situatedness in design

Situatedness in design is concerned with learning the situation-dependent application conditions of design knowledge. According to Reffat and Gero, situatedness means that the actions taken in a design process are a function of both the design state and the way this state is interpreted (as part of maintaining a design strategy) [Reffat and Gero, 2000]. Therefore, learning the applicability conditions of design knowledge enables to locate design knowledge that is relevant to the current state of a design process.

For example, Reffat and Gero present a computational system of situated learning in design, called SLiDe. Applied to the domain of architectural design, SLiDe learns about the applicability conditions of architectural design knowledge by capturing the regularities of relevant relationships among architectural shapes across different design states. The result of this learning effort is that SLiDe is able to locate shapes that are relevant to the design situation at hand.

For further research on situatedness in design, the same directions as given earlier for learning in design apply. The research should address situatedness in relation to design object description manipulation (as in the SLiDe example), requirement qualification set manipulation (e.g., to learn about the relationships between needs and desires on one hand and design requirements operationalising these needs and desires on the other hand), and design process co-ordination (e.g., to learn about the relationships between design problems on one hand and design strategies for design processes faced with these design problems on the other hand).

16.1.1.4 Distributed design

Distributed design as a subject of research is a popular topic. Enabled by the recent technological advances made (especially the Internet), distributed design is rapidly becoming an essential part of the development of complex (consumer) products. The term denotes a process in which multiple agents (e.g., customers, designers and production workers) co-operate to negotiate requirements and to produce a satisfactory design, but not necessarily at the same time or at the same location.

An agent is an entity that exhibits reactive, pro-active, social and autonomous behaviour ([Wooldridge and Jennings, 1995]). By studying the ways that design agents interact, theories and models of co-operation can be developed that pave the ground for the development of multi-agent design (support) systems with advanced human-computer interaction. Before introducing research directions related to the subject of distributed design, first some results of past research on multi-agent design are presented that give an idea of what the use might be of distinguishing different agents in a design process.

Grecu and Brown describe a system for parametric spring design, built from small knowledge-based systems called Single Function Agents (SiFAs) [Grecu and Brown, 1996]. Each SiFA is deliberately constructed to have restricted capabilities only; therefore, the agent must interact with other SiFAs that have other capabilities. Only by co-operation, they are able to carry out a design task. There are different types of SiFAs, each having a single target, which is to select, estimate, evaluate, criticise or praise parameter values. Due to their ability to learn, the SiFAs are capable of reducing the number of conflicts during design.

Campbell, Cagan and Kotovsky present a theory of engineering design, A-design, that models an engineering design process as a complex adaptive system of interacting software agents [Campbell, Cagan and Kotovsky, 1998]. According to this theory, *configuration agents* create conceptual designs, which are filled with actual components from a catalogue by *instantiation agents*. After evaluation, *fragment agents* remove 'bad' components from the design, whereas subsystem agents extract 'good' sub-assemblies (consisting of multiple components) and store it in the catalogue for use in future iterations of the design process or for later use in a new design process. These four types of agent are maintained by *manager agents*, that adjust the number of agents of a given type on the basis of their contribution to 'good' and 'bad' designs. Thus, agents with successful contributions outlive agents without successful contributions, which should yield more successful design processes.

McAlinden, Florida-James, Chao, Norman, Hills and Smith [ibid., 1998] describe how design agents can be integrated to facilitate information and knowledge sharing, using a central product model of the STEP standard as well as ACL (Agent Communication Language) and knowledge-based ontologies. They claim that their approach ensures that new, existing and legacy design systems can be used right away (i.e., without delay) in a design project. (However, as Vanwelkenhuysen and Mizoguchi point out, straightforward knowledge reuse may not always be feasible, since developing an ontology completely independent of the purpose for which it is used is not possible [Vanwelkenhuysen and Mizoguchi, 1995]).

From a knowledge-level perspective, the development of theories and models for distributed design means that the generic process composition, knowledge composition and relation between these compositions has to be determined for distributed design processes. For this purpose, this thesis can be used as a point of departure, besides other design theories such as Smithers' knowledge-level theory of design processes [Smithers, 1998]. Furthermore, as agents need to co-operate to complete a distributed design project successfully, multi-agent models of project management are indispensable ([Brazier, Cornelissen, Jonker and Treur, 2000]).

This research may endorse the results of past design research in which agents were not considered. Firstly, it may validate the original theories and models as adequate for design tasks of individual agents. Secondly, the research could result in a mapping between single-agent theories and models of design processes and theories and models of distributed design processes.

16.1.2 Generic design model

At least two subjects directly related to the generic design model GDM itself are worthy of further research. Firstly, as pointed out in Chapter 15, with GDM relatively little attention has been paid to modelling the process of design process co-ordination within a design process (compared to the process of requirement qualification set manipulation and the process of design object description manipulation). A logical analysis of design process co-ordination processes could be expected to yield valuable knowledge about how strategic decision making affect the course of design processes and vice versa.

Secondly, further research is required to develop specialisations of GDM that are useful in relation to specific design methods and specific application domains. Chapter 9 provided some examples, but many other specialisations exist, which may be equally (or even more) useful.

16.2 Design Methods

As pointed out in Chapter 15, this thesis has paid no attention to the issue of representing the design problem space, the design solution space and the design knowledge for a given application domain. Chapter 15 suggested that it would be helpful to have a set of guidelines that describe which types of representations would fit best which (specialised) parts of GDM.

This means that research will have to focus on new analysis techniques for the determination of the process composition, knowledge composition and relation between process composition and knowledge composition of a specific design method, given the representations it employs. It also means that research will have to focus on new synthesis techniques for the generation of suitable representations to be employed in a specific design method, given its process composition, knowledge composition and relation between process composition and knowledge composition.

16.3 Design Support Systems

With GDM as a generic model of design processes and DESIRE as a compositional multi-agent development method, a well-founded basis for the (conceptual) development of design support systems is provided. As pointed out in Chapter 15, what is missing is a design application development environment for a developer of a design support system. Such an environment would have to contain the following facilities.

- **Requirements engineering.** These facilities should support the construction of requirement models representing the user's design requirements. Besides the facilities developed by the AI in Design community (e.g., [Sumi, Hori and Ohsuga, 1998]), also the facilities distinguished by the software engineering community (e.g., [Davis,

1993; Sommerville and Sawyer, 1997]) and the agents community (e.g., [Herlea, Jonker, Treur and Wijngaards, 1999*a*; Herlea, Jonker, Treur and Wijngaards, 1999*b*]) are worthwhile as part of a design application development environment.

- **Creation of (shared) models of design.** These facilities should support the creation of models of design processes, such that they can be shared between different design agents (e.g., [Brazier, Jonker, Treur and Wijngaards, 2000]). Such a model should be formulated in a knowledge level language such as the one provided by DESIRE (see Chapter 5), CommonKADS ([Schreiber, Wielinga, Hoog, Akkermans and Van de Velde, 1996]), or KARL (Fensel, Angele and Landes, 1991; Fensel, 1993; Landes, 1994]). The facilities should further include a library with reusable components (e.g., [Breuker, 1994; Campbell, Cagan and Kotovsky, 1998]) and should support the specification, validation and verification of models of compositional design support systems (e.g., [Jonker and Treur, 1998; Cornelissen, Jonker and Treur, 2002]).

- **Operationalisation of design support system models.** Operationalisation should include user interfacing and knowledge representation (e.g., [Candy, 1997]). The representations used should take different forms according to the domain context, the intended users and their level of expertise, and the intended purpose of using the representations (e.g., [Candy, 1998]). Especially creativity will be a challenging subject of operationalising a design support system model (e.g., [Candy and Edmonds, 1994; Candy, Edmonds and Patrick, 1995; Candy and Edmonds, 1996]).

Closely related to design support is creativity support, as it plays a significant role in most design processes. For example, Fischer and Nakakoji describe a conceptual and technical framework for creativity supported by domain-oriented design environments [Fischer and Nakakoji, 1997]. Another example is a creativity support system named En Passant 2, developed by Aihara and Hori, which stores the user's research notes and gives triggers in the current context to recall relevant notes from memory [Aihara and Hori, 1998]. Such research can be used to advantage in further research on design support systems.

Appendix A

Partial Semantics of Order-Sorted Predicate Logic

This appendix forms an extension to Chapter 3 of this thesis, and defines the partial semantics of order-sorted predicate logic. It can be read instead of Definitions 3.3.1 to 3.3.6 in Section 3.3, which for ease of reading and understanding are restricted to the partial semantics of proposition logic.

Publications. *This appendix is based on earlier research of Van Langen and Treur on the representation of information states by means of partial models [Langen and Treur, 1989].*

The partiality of the semantics of predicate-logical sentences in an information state may appear in different ways. Blamey distinguishes the following cases ([Blamey, 1986]):

- a term t may lack a denotation and may, for this reason, make a sentence $\phi(t)$ neither true nor false;
- a predicate $\phi(x)$ may not be defined as either true or false for all objects so that, if t denoted such an object, $\phi(t)$ would be neither true nor false.

For partial terms, the presence of a distinguished object '$\otimes$' is assumed, reserved to be the interpretation of a 'non-denoting' term. For a partial predicate p, two different sets are used, p^+ and p^-; when p is undefined, its interpretation does not belong to either one of these sets. Such an interpretation is possible in a three-valued logic, which distinguishes *true* (denoted 1), *false* (denoted 0), and a third truth value, *undefined* (denoted u). The purpose of the non-classical truth value *undefined* is to indicate a state of partial ignorance about the truth value of a sentence.

Definition A.1. (Partial Model) Given an order-sorted signature $\Sigma = (\mathbf{S}, \mathbf{C}, \mathbf{F}, \mathbf{P})$, a *partial Σ-model M* consists of

1. A non-empty set D_M called the *domain.*

2. For each sort $S \in \mathbf{S}$, a set $S_M \subseteq D_M$, such that $\top_M = D_M$, $\bot_M = \emptyset$, and for all $S \in \mathbf{S} \setminus \{\bot\}$, $S_M \neq \emptyset$. For every $S_1, S_2 \in \mathbf{S}$, if $S_1 \leq S_2$, then $(S_1)_M \subseteq (S_2)_M$.

3. For each constant c: S, an element $c_M \in S_M \cup \{\otimes\}$.

4. For each function f: $S_1 \ldots S_n \rightarrow S$, a function f_M: $(S_1)_M \cup \{\otimes\} \times \ldots \times (S_n)_M \cup \{\otimes\} \rightarrow S_M \cup \{\otimes\}$.

5. For each predicate p: $S_1 \ldots S_n$, two disjoint relations p^+_M: $(S_1)_M \cup \{\otimes\} \times \ldots \times (S_n)_M \cup \{\otimes\}$ and p^-_M: $(S_1)_M \cup \{\otimes\} \times \ldots \times (S_n)_M \cup \{\otimes\}$.

To be able to interpret quantified sentences (i.e., sentences of the form $(\forall x\text{: } S)\ (\varphi)$ or of the form $(\exists x\text{: } S)\ (\varphi)$), functions are needed that map the variables in such sentences onto objects of the domain. Such functions are called assignments.

Definition A.2. (Assignment) Given an **S**-indexed family of variables $V = \{V_S \mid S \in \mathbf{S}\}$, a family of *(sorted) assignments* to V is an **S**-indexed family of functions $A = \{a_S\text{: } V_S \rightarrow S_M \cup \{\otimes\} \mid S \in \mathbf{S}\}$.

Given an assignment a_S, a variable x: S and a domain object d in the domain of a partial model, let $a_S(x|d)$ be the assignment such that:

- $a_S(x|d)(x) = d$ and
- $a_S(x|d)(y) = a_S(y)$ if y: S is a variable distinct from x.

In addition, let $\mathbb{T} = \{0, 1, u\}$ be a set of *truth values*, where 0 denotes the truth value *false*, 1 denotes *true* and u denotes *undefined*. Then the partial semantics of terms and sentences is defined as follows.

Definition A.3. (Partial Semantics) Let $\Sigma = (\mathbf{S}, \mathbf{C}, \mathbf{F}, \mathbf{P})$ be a signature, M a partial Σ-model, V an **S**-indexed family of sorted variables, and A an **S**-indexed family of assignments to V. Then:

1. $M_A(x) = a_S(x)$ for all variables x: S;
   $M_A(c) = c_M$ for all constants c: S;
   $M_A(f(t_1, \ldots, t_n)) = f_M(M_A(t_1), \ldots, M_A(t_n))$ for all functions f: $S_1 \times \ldots \times S_n \rightarrow S$.

2. $M_A(t_1 = t_2) = 1$ iff $M_A(t_1) \in D_M$, $M_A(t_2) \in D_M$, and $M_A(t_1)$ is identical to $M_A(t_2)$;
   $M_A(t_1 = t_2) = 0$ iff $M_A(t_1) \in D_M$, $M_A(t_2) \in D_M$, and $M_A(t_1)$ is not identical to $M_A(t_2)$;
   $M_A(t_1 = t_2) = u$ otherwise (i.e., iff $M_A(t_1) = \otimes$ or $M_A(t_2) = \otimes$).

3. $M_A(p(t_1, \ldots, t_n)) = 1$ iff $M_A(t_1) \in D_M$, $\ldots M_A(t_n) \in D_M$, and $p^+_M(M_A(t_1), \ldots, M_A(t_n))$;
   $M_A(p(t_1, \ldots, t_n)) = 0$ iff $M_A(t_1) \in D_M$, $\ldots M_A(t_n) \in D_M$, and $p^-_M(M_A(t_1), \ldots, M_A(t_n))$;
   $M_A(p(t_1, \ldots, t_n)) = u$ otherwise (i.e., iff $M_A(t_1) = \otimes$ or $\ldots$ or $M_A(t_n) = \otimes$).

4. If $\varphi, \phi \in \text{WFF}(\Sigma)$, then:
   $M_A(\neg\, \varphi) = 1$ iff $M_A(\varphi) = 0$,
   $M_A(\neg\, \varphi) = 0$ iff $M_A(\varphi) = 1$,
   $M_A(\neg\, \varphi) = u$ otherwise;

   $M_A(\varphi \wedge \phi) = 1$ iff $M_A(\varphi) = 1$ and $M_A(\phi) = 1$,
   $M_A(\varphi \wedge \phi) = 0$ iff $M_A(\varphi) = 0$ or $M_A(\phi) = 0$,
   $M_A(\varphi \wedge \phi) = u$ otherwise;

   $M_A(\varphi \vee \phi) = 1$ iff $M_A(\varphi) = 1$ or $M_A(\phi) = 1$,
   $M_A(\varphi \vee \phi) = 0$ iff $M_A(\varphi) = 0$ and $M_A(\phi) = 0$,
   $M_A(\varphi \vee \phi) = u$ otherwise;

   $M_A(\varphi \Rightarrow \phi) = 1$ iff $M_A(\varphi) = 0$ or $M_A(\phi) = 1$,
   $M_A(\varphi \Rightarrow \phi) = 0$ iff $M_A(\varphi) = 1$ and $M_A(\phi) = 0$,
   $M_A(\varphi \Rightarrow \phi) = u$ otherwise;

   $M_A(\varphi \Leftrightarrow \phi) = 1$ iff either $M_A(\varphi) = 1$ and $M_A(\phi) = 1$, or $M_A(\varphi) = 0$ and $M_A(\phi) = 0$,
   $M_A(\varphi \Leftrightarrow \phi) = 0$ iff either $M_A(\varphi) = 1$ and $M_A(\phi) = 0$, or $M_A(\varphi) = 0$ and $M_A(\phi) = 1$,
   $M_A(\varphi \Leftrightarrow \phi) = u$ otherwise.

5. If $\varphi \in \text{WFF}(\Sigma)$, then:
   $M_A((\forall x\colon S)\ (\varphi)) = 1$ iff M_A fully interprets S and $M_{A(x|d)}(\varphi) = 1$ for all $d \in S_M$,
   $M_A((\forall x\colon S)\ (\varphi)) = 0$ iff $M_{A(x|d)}(\varphi) = 0$ for some $d \in S_M$,
   $M_A((\forall x\colon S)\ (\varphi)) = u$ otherwise;

   $M_A((\exists x\colon S)\ (\varphi)) = 1$ iff $M_{A(x|d)}(\varphi) = 1$ for some $d \in S_M$,
   $M_A((\exists x\colon S)\ (\varphi)) = 0$ iff M_A fully interprets S and $M_{A(x|d)}(\varphi) = 0$ for all $d \in S_M$,
   $M_A((\exists x\colon S)\ (\varphi)) = u$ otherwise.

Note that a partial model M is said to *fully interpret* a sort S under a family of assignments A if for each well-formed term t of sort S, it holds that $M_A(t) \neq \otimes$.

Definition A.4. (Truth and Completeness) Let Σ be a signature, M a partial Σ-model, and A a family of assignments. A sentence $\varphi \in \text{WFF}(\Sigma)$ is *true* in M under A if $M_A(\varphi) = 1$, it is *false* in M under A if $M_A(\varphi) = 0$, and it is *undefined* in M under A if $M_A(\varphi) = u$. Furthermore, M is *complete* under A if (1) for every well-formed term t it holds that $M_A(t) \neq \otimes$, and (2) if every sentence is either true or false in M under A.

The notion of a complete partial model corresponds to the classical notion of a structure in standard logics.

Definition A.5. (Satisfaction) Let Σ be a signature, M a partial Σ-model, V an **S**-indexed family of sorted variables, and A an **S**-indexed family of assignments to V. Then the *satisfaction* relation $\models$ is defined for all $\varphi \in \text{WFF}(\Sigma)$ as follows:

$M_A \models \varphi$ if $M_A(\varphi) = 1$; $M_A \not\models \varphi$ if $M_A(\varphi) = 0$.

Definition A.6. (Model) Let Σ be a signature, and $\varphi \in \text{WFF}(\Sigma)$ a well-formed formula. Then a partial Σ-model M is a *model* of φ, denoted $M \models \varphi$, if $M_A \models \varphi$ for each family of assignments A. For a subset $\Phi \subset \text{WFF}(\Sigma)$, $M \models \Phi$ denotes that M is a model of each element of Φ.

In this thesis, only sentences are considered, rather than formulas with free variables. For the purpose of this thesis, this assumption is not restrictive and it has the advantage that for the semantics, it does not matter which particular assignment is used. This follows from the property that if a sentence is true under one family of assignments, then it is true under any family of assignments ([Blamey, 1986]).

Appendix B

Textual Specification of GDM in DESIRE

This appendix presents a textual specification of the generic design model GDM, expressed in the specification language that is part of DESIRE.

The textual specification of the generic design model GDM below is expressed in the knowledge-level specification language that is part of the component-based multi-agent design method DESIRE. Refer to Chapter 5 for a brief overview of DESIRE and to Chapters 6, 7 and 8 for an explanation of GDM.

```
component GDM
  task information
    public task information
    private task information
      components
        design;
      initial task information
        extent all_p;
      task control contents
        knowledge bases
          empty_task_control_kbs;

  kernel information
    public kernel information
      knowledge structures
```

```
information type domain_object_type
  sorts
    domain_object_variable, domain_object_constant, domain_object
  subsorts
    domain_object_variable, domain_object_constant: domain_object;
  objects
    ExampleVariable: domain_object_variable;
    ExampleConstant: domain_object_constant;
end information type

information type attribute_type
  sorts
    attribute
  objects
    ExampleAttribute: attribute;
end information type

information type value_type
  information types
    domain_object_type;
  sorts
    domain_object_list, value
  subsorts
    domain_object: domain_object_list;
    domain_object_list: value;
  objects
    nil: domain_object_list;
    ExampleValue: value;
  functions
    dot: domain_object * domain_object_list -> domain_object_list;
end information type

information type general_domain_object_information
  information types
    domain_object_type, attribute_type, value_type;
  relations
    has_value: domain_object * attribute * value;
    has_part: domain_object * domain_object;
end information type
```

```
information type application_specific_domain_object_information
  information types
    domain_object_type;
end information type

information type domain_object_information
  information types
    general_domain_object_information,
    application_specific_domain_object_information;
end information type

information type domain_object_info_element_type
  sorts
    domain_object_info_element
  meta-descriptions
    domain_object_information : domain_object_info_element;
end information type

information type DOD_name_type
  sorts
    DOD_name
  objects
    EmptyDOD, ExampleName: DOD_name;
end information type

information type sign_type
  sorts
    sign
  objects
    pos, neg, unk: sign;
end information type

information type DOD
  information types
    DOD_name_type, domain_object_info_element_type, sign_type;
  relations
    includes_domain_object_information:
      DOD_name * domain_object_info_element * sign;
end information type
```

```
information type domain_object_info_formula_type
  information types
    domain_object_info_element_type;
  sorts
    domain_object_info_formula
  subsorts
    domain_object_info_element: domain_object_info_formula;
  functions
    l_not: domain_object_info_formula -> domain_object_info_formula;
    l_or, l_and, l_implies: domain_object_info_formula * domain_object_info_formula ->
      domain_object_info_formula;
    exists, for_all: domain_object_variable * domain_object_info_formula ->
      domain_object_info_formula;
end information type

information type requirement_type
  information types
    domain_object_info_formula_type;
  sorts
    requirement_name, requirement
  subsorts
    requirement_name, domain_object_info_formula: requirement;
  objects
    ExampleName: requirement_name;
end information type

information type requirement_definitions
  information types
    requirement_type;
  relations
    is_defined_as: requirement_name * domain_object_info_formula;
end information type

information type numeric_constraint_type
  sorts
    integer, numeric_constraint
  objects
    zero: integer;
    any, all_possible: numeric_constraint;
  functions
    succ: integer -> integer;
```

```
    at_least, exactly, at_most: integer -> numeric_constraint;
end information type

information type qualification_type
  information types
    numeric_constraint_type;
  sorts
    qualification
  objects
    every: qualification;
  functions
    at_random, in_preferred_order: numeric_constraint -> qualification;
end information type

information type qualified_requirement_type
  information types
    requirement_type,
    qualification_type;
  sorts
    qualified_requirement_name, requirement_list,
    qualified_requirement_expression, qualified_requirement
  subsorts
    requirement: requirement_list;
    qualified_requirement_name,
    qualified_requirement_expression: qualified_requirement;
  objects
    ExampleName: qualified_requirement_name;
    nil: requirement_list;
  functions
    dot: requirement * requirement_list -> requirement_list;
    expr: qualification * requirement_list -> qualified_requirement_expression;
end information type

information type qualified_requirement_definitions
  information types
    qualified_requirement_type;
  relations
    is_defined_as: qualified_requirement_name * qualified_requirement_expression;
end information type
```

```
information type design_requirement_definitions
  information types
    requirement_definitions,
    qualified_requirement_definitions;
end information type

information type design_requirement_type
  information types
    requirement_type, qualified_requirement_type;
  sorts
    design_requirement
  subsorts
    requirement, qualified_requirement: design_requirement;
end information type

information type design_requirement_enactment_information
  information types
    design_requirement_type;
  relations
    is_to_be_satisfied: design_requirement;
end information type

information type application_specific_design_requirement_information
  information types
    design_requirement_type;
end information type

information type design_requirement_information
  information types
    design_requirement_definitions,
    design_requirement_enactment_information,
    application_specific_design_requirement_information;
end information type

information type design_requirement_list_type
  information types
    design_requirement_type;
  sorts
    design_requirement_list
  subsorts
    design_requirement: design_requirement_list;
```

```
  objects
    nil: design_requirement_list;
  functions
    dot: design_requirement * design_requirement_list -> design_requirement_list;
end information type

information type basic_DOD_assessments
  information types
    DOD_name_type,
    design_requirement_list_type;
  relations
    satisfies,
    violates,
    can_be_refined_to_satisfy,
    is_decisive_wrt_satisfaction_of: DOD_name * design_requirement_list;
    is_refinement_of, is_consistent_with: DOD_name * DOD_name;
end information type

information type design_requirement_assessments
  information types
    design_requirement_list_type;
  relations
    is_satisfiable, is_tautological, is_contradictory, is_precise: design_requirement_list;
end information type

information type design_requirement_info_element_type
  sorts
    design_requirement_info_element
  meta-descriptions
    design_requirement_information : design_requirement_info_element;
end information type

information type RQS_name_type
  sorts
    RQS_name
  objects
    EmptyRQS, ExampleName: RQS_name;
end information type
```

information type RQS
  information types
    RQS_name_type,
    design_requirement_info_element_type,
    sign_type;
  relations
    includes_design_requirement_information:
      RQS_name * design_requirement_info_element * sign;
end information type

information type overall_DOD_assessments
  information types
    DOD_name_type, RQS_name_type;
  relations
    fulfils, fails_to_fulfil, can_be_refined_to_fulfil, is_decisive_wrt_fulfilment_of:
      DOD_name * RQS_name;
end information type

information type overall_RQS_assessments
  information types
    RQS_name_type;
  relations
    can_be_fulfilled, is_inconsistent, is_ambiguous, is_imprecise, is_incomplete:
      RQS_name;
end information type

information type basic_evaluation_info_element_type
  sorts
    basic_evaluation_info_element
  meta-descriptions
    basic_DOD_assessments, design_requirement_assessments :
      basic_evaluation_info_element;
end information type

information type RQS_specific_basic_evaluation_information
  information types
    RQS_name_type, basic_evaluation_info_element_type, sign_type;
  relations
    includes_basic_evaluation_information:
      RQS_name * basic_evaluation_info_element * sign;
end information type

information type DOD_assessments
  information types
    RQS_specific_basic_evaluation_information,
    overall_DOD_assessments;
end information type

information type RQS_assessments
  information types
    RQS_specific_basic_evaluation_information,
    overall_RQS_assessments;
end information type

information type DOD_solution_information
  information types
    DOD_name_type, RQS_name_type;
  relations
    is_basis_reduct_of: DOD_name * DOD_name;
    is_DOD_solution_to: DOD_name * RQS_name;
end information type

information type RQS_solution_information
  information types
    RQS_name_type;
  relations
    is_acceptable_substitute_for: RQS_name * RQS_name;
    is_RQS_solution_to: RQS_name * RQS_name;
end information type

information type process_status_type
  sorts
    process_status
  objects
    idle, active: process_status
end information type

information type process_status_information
  information types
    process_status_type;
  relations
    is_previous_status, is_current_status, is_next_status: process_status;
end information type

```
information type design_resource_type
  sorts
    design_resource
  objects
    ExampleResource: design_resource;
end information type

information type design_resource_consumption_information
  information types
    design_resource_type, numeric_constraint_type;
  relations
    has_past_consumption_of, has_future_consumption_of:
      design_resource * numeric_constraint;
end information type

information type application_specific_design_process_results_information
  information types
    DOD_name_type, RQS_name_type, process_status_type, design_resource_type;
end information type

information type design_process_results_information
  information types
    RQS_solution_information,
    DOD_solution_information,
    process_status_information,
    design_resource_consumption_information,
    application_specific_design_process_results_information;
end information type

information type design_process_results_info_element_type
  sorts
    design_process_results_info_element
  meta-descriptions
    design_process_results_information : design_process_results_info_element;
end information type

information type design_process_results_info_formula_type
  information types
    design_process_results_info_element_type;
  sorts
    design_process_results_info_formula
```

```
  subsorts
    design_process_results_info_element: design_process_results_info_formula;
  functions
    l_not: design_process_results_info_formula -> design_process_results_info_formula;
    l_or, l_and, l_implies:
     design_process_results_info_formula * design_process_results_info_formula ->
     design_process_results_info_formula;
end information type

information type process_objective_type
  information types
    design_process_results_info_formula_type;
  sorts
    process_objective_name, process_objective
  subsorts
    process_objective_name, design_process_results_info_formula: process_objective;
  objects
    ExampleName: process_objective_name;
end information type

information type process_objective_definitions
  information types
    process_objective_type;
  relations
    is_defined_as: process_objective_name * design_process_results_info_formula;
end information type

information type qualified_process_objective_type
  information types
    process_objective_type,
    qualification_type;
  sorts
    qualified_process_objective_name, process_objective_list,
    qualified_process_objective_expression, qualified_process_objective
  subsorts
    process_objective: process_objective_list;
    qualified_process_objective_name,
    qualified_process_objective_expression: qualified_process_objective;
  objects
    ExampleName: qualified_process_objective_name;
    nil: process_objective_list;
```

```
  functions
    dot: process_objective * process_objective_list -> process_objective_list;
    expr: qualification * process_objective_list -> qualified_process_objective_expression;
end information type

information type qualified_process_objective_definitions
  information types
    qualified_process_objective_type;
  relations
    is_defined_as:
     qualified_process_objective_name * qualified_process_objective_expression;
end information type

information type design_process_objective_definitions
  information types
    process_objective_definitions,
    qualified_process_objective_definitions;
end information type

information type design_process_objective_type
  information types
    process_objective_type, qualified_process_objective_type;
  sorts
    design_process_objective
  subsorts
    process_objective, qualified_process_objective: design_process_objective;
end information type

information type design_process_objective_enactment_information
  information types
    design_process_objective_type;
  relations
    is_to_be_satisfied: design_process_objective;
end information type

information type application_specific_design_process_objective_information
  information types
    design_process_objective_type;
end information type
```

```
        information type design_process_objectives
          information types
            design_process_objective_definitions,
            design_process_objective_enactment_information,
            application_specific_design_process_objective_information;
        end information type

        information type epistemic_design_process_performance_information
          information types
            design_process_results_info_element_type, sign_type;
          relations
            is_part_of_design_process_results: design_process_results_info_element * sign;
        end information type

        information type design_process_objective_evaluations
          information types
            design_process_objective_type;
          relations
            is_satisfied, is_violated, is_decided: design_process_objective;
        end information type

        information type design_process_evaluations
          information types
            epistemic_design_process_performance_information,
            design_process_objective_evaluations;
        end information type

    private kernel information
component design
  task information
    public task information
      task control foci
        main;
      evaluation criteria
        main;
    private task information
      components
        DPC,
        RQSM,
        DODM;
```

```
  information links
    given_design_process_objectives,
    given_RQS,
    given_DOD,
    intermediate_overall_design_strategy_to_RQSM,
    intermediate_overall_design_strategy_to_DODM;
    intermediate_RQSM_process_evaluations,
    intermediate_DODM_process_evaluations,
    intermediate_RQS,
    intermediate_DOD_assessments,
    resulting_design_process_evaluations,
    resulting_RQS,
    resulting_RQS_assessments,
    resulting_DOD,
    resulting_DOD_assessments,
  initial task information
    task control focus main;
    extent all_p;
  task control contents
knowledge base design_task_control
  information types design_task_control_sig
  contents
    if start
    then next_component_state(DPC, active)
     and next_link_state(given_design_process_objectives, uptodate)
     and next_link_state(given_RQS, uptodate)
     and next_link_state(given_DOD, uptodate);

    if previous_component_state(DPC, active)
     and component_state(DPC, idle)
     and evaluation(DPC, main, any_new, succeeded)
    then next_component_state(RQSM, active)
     and next_component_state(DODM, active)
     and next_link_state(intermediate_overall_design_strategy_to_RQSM, uptodate)
     and next_link_state(intermediate_overall_design_strategy_to_DODM, uptodate);

    if previous_component_state(RQSM, active)
     and component_state(RQSM, idle)
     and component_state(DODM, idle)
```

```
        then next_component_state(DPC, active)
         and next_link_state(intermediate_RQSM_process_evaluations, uptodate)
         and next_link_state(intermediate_RQS, uptodate)
         and next_link_state(intermediate_DODM_process_evaluations, uptodate)
         and next_link_state(intermediate_DOD_assessments, uptodate);

        if previous_component_state(DODM, active)
         and component_state(DODM, idle)
         and component_state(RQSM, idle)
        then next_component_state(DPC, active)
         and next_link_state(intermediate_RQSM_process_evaluations, uptodate)
         and next_link_state(intermediate_RQS, uptodate)
         and next_link_state(intermediate_DODM_process_evaluations, uptodate)
         and next_link_state(intermediate_DOD_assessments, uptodate);

        if previous_component_state(DPC, active)
         and component_state(DPC, idle)
         and evaluation(DPC, main, any_new, failed)
        then next_link_state(resulting_design_process_evaluations, uptodate)
         and next_link_state(resulting_RQS_assessments, uptodate)
         and next_link_state(resulting_RQS, uptodate)
         and next_link_state(resulting_DOD_assessments, uptodate)
         and next_link_state(resulting_DOD, uptodate)
         and stop;
    end knowledge base /* design_task_control */

  kernel information
    public kernel information
      knowledge structures
        information type selection_criterion_type
          sorts
            selection_criterion
          objects
            is_true, is_false, ExampleCriterion: selection_criterion;
        end information type

        information type selection_criterion_list_type
          information types
            selection_criterion_type;
          sorts
            selection_criterion_list
```

```
  subsorts
    selection_criterion: selection_criterion_list;
  objects
    nil: selection_criterion_list;
  functions
    dot: selection_criterion * selection_criterion_list -> selection_criterion_list;
end information type

information type control_process_plan_type
  information types
    RQS_name_type, DOD_name_type,
    selection_criterion_list_type, design_process_results_info_formula_type;
  sorts
    control_approach, control_process_plan
  objects
    transformation, translation,
    decomposition, composition,
    reduction, extension: control_approach;
  functions
    continue_with: RQS_name * DOD_name -> control_process_plan;
    if_then: design_process_results_info_formula * control_process_plan ->
      control_process_plan;
    if_then_else: design_process_results_info_formula * control_process_plan *
      control_process_plan -> control_process_plan;
    while_do: design_process_results_info_formula * control_process_plan ->
      control_process_plan;
    repeat_until: control_process_plan * design_process_results_info_formula ->
      control_process_plan;
    do_in_sequence, do_in_parallel:
      control_process_plan * control_process_plan ->
      control_process_plan;
    apply_criteria_for_retrieval_approach, apply_criteria_for_modification_approach:
      selection_criterion_list * control_approach -> control_process_plan;
end information type

information type design_strategy_type
  information types
    control_process_plan_type;
  sorts
    design_strategy_name, design_strategy
```

```
  subsorts
    design_strategy_name,
    control_process_plan: design_strategy;
  objects
    ExampleName: design_strategy_name;
end information type

information type design_strategy_definitions
  information types
    design_strategy_type;
  relations
    is_defined_as: design_strategy_name * control_process_plan;
end information type

information type design_process_state_type
  sorts
    design_process_state
  objects
    InitialState, ExampleState: design_process_state;
  functions
    succ: design_process_state -> design_process_state;
end information type

information type current_overall_design_process_information
  information types
    design_process_state_type,
    control_process_plan_type;
  relations
    is_current_overall_design_process_state: design_process_state;
    is_current_control_process_plan: control_process_plan;
end information type

information type state_specific_design_strategy_information
  information types
    design_process_state_type,
    design_strategy_type;
  relations
    includes_design_strategy: design_process_state * design_strategy;
end information type
```

```
information type overall_design_strategy
  information types
    design_strategy_definitions,
    current_overall_design_process_information,
    state_specific_design_strategy_information;
end information type

information type control_process_evaluation_type
  sorts
    control_process_evaluation
  objects
    incomplete, succeeded, failed: control_process_evaluation
end information type

information type DODM_process_evaluations
  information types
    design_strategy_type, control_process_evaluation_type;
  relations
    has_DODM_process_evaluation: design_strategy * control_process_evaluation;
end information type

information type RQSM_process_evaluations
  information types
    design_strategy_type, control_process_evaluation_type;
  relations
    has_RQSM_process_evaluation: design_strategy * control_process_evaluation;
end information type

information type control_process_evaluations
  information types
    RQSM_process_evaluations,
    DODM_process_evaluations;
end information type

information type current_control_process_state_identity_information
  information types
    design_process_state_type;
  relations
    is_current_control_process_state: design_process_state;
end information type
```

```
information type state_specific_design_process_results_information
  information types
    design_process_state_type, design_process_results_info_element_type, sign_type;
  relations
    includes_design_process_results_information:
     design_process_state * design_process_results_info_element * sign;
end information type

information type design_process_state_sequences
  information types
    design_process_state_type;
  relations
    has_succeeding_design_process_state:
      design_process_state * design_process_state;
end information type

information type control_decision_type
  sorts
    control_decision
  objects
    termination, replacement, modification,
    deductive_refinement, query_and_retrieval: control_decision
end information type

information type state_specific_control_decision_information
  information types
    design_process_state_type,
    control_decision_type;
  relations
    includes_control_decision: design_process_state * control_decision;
end information type

information type state_specific_control_process_evaluations
  information types
    design_process_state_type,
    control_process_evaluation_type;
  relations
    includes_control_process_evaluation:
     design_process_state * control_process_evaluation;
end information type
```

```
information type manipulation_trace_information
  information types
    design_strategy_definitions,
    state_specific_design_strategy_information,
    current_control_process_state_identity_information,
    state_specific_design_process_results_information,
    design_process_state_sequences,
    state_specific_control_decision_information,
    state_specific_control_process_evaluations;
end information type

information type current_control_decision_information
  information types
    control_decision_type;
  relations
    is_current_control_decision: control_decision;
end information type

information type state_specific_design_strategy_information_queries
  information types
    design_process_state_type,
    design_strategy_type;
  relations
    is_included_in_which_design_process_states: design_strategy;
    includes_which_design_strategy: design_process_state;
end information type

information type state_specific_design_process_results_information_queries
  information types
    design_process_state_type,
    design_process_results_info_element_type,
    sign_type;
  relations
    is_included_in_which_design_process_states:
     design_process_results_info_element * sign;
    includes_which_design_process_results_information: design_process_state;
end information type

information type design_process_state_sequence_queries
  information types
    design_process_state_type;
```

```
  relations
    is_preceded_by_which_design_process_state,
    is_succeeded_by_which_design_process_state: design_process_state;
end information type

information type state_specific_control_decision_information_queries
  information types
    design_process_state_type,
    control_decision_type;
  relations
    includes_which_control_decision: design_process_state;
    is_decision_in_which_design_process_states: control_decision;
end information type

information type state_specific_control_process_evaluation_queries
  information types
    control_process_evaluation_type,
    design_process_state_type;
  relations
    is_evaluation_of_which_design_process_states: control_process_evaluation;
    includes_which_control_process_evaluation: design_process_state;
end information type

information type manipulation_trace_information_queries
  information types
    state_specific_design_strategy_information_queries,
    state_specific_design_process_results_information_queries,
    design_process_state_sequence_queries,
    state_specific_control_decision_information_queries,
    state_specific_control_process_evaluation_queries;
end information type

information type current_DOD_identity_information
  information types
    DOD_name_type;
  relations
    is_current_DOD: DOD_name;
end information type
```

```
        information type current_RQS_identity_information
          information types
            RQS_name_type;
          relations
            is_current_RQS: RQS_name;
        end information type

      public levels
        level_1 , level_2 , level_3;
      public level chain
        level_1 < level_2 < level_3;

      input interface
        level level_1
          information type DOD;
        level level_2
          information type RQS;
        level level_3
          information type design_process_objectives;

      output interface
        level level_1
          information type DOD;
        level level_2
          information type design_2nd_meta_level_output
            information types
              RQS_assessments,
              DOD_assessments,
              RQS;
          end information type
        level level_3
          information type design_process_evaluations;

    private kernel information
component DPC
  task information
    public task information
      task control foci
        main;
      evaluation criteria
        main;
```

```
    private task information
      initial task information
        task control focus main;
        extent all_p;

      task control contents
        knowledge bases
          empty_task_control_kbs;

  kernel information
    public kernel information
      knowledge structures
      public levels
        level_1;
      public level chain
        level_1;

      input interface
        level level_1
          information type DPC_3rd_meta_level_input
            information types
              design_process_objectives,
              control_process_evaluations;
          end information type

      output interface
        level level_1
          information type DPC_3rd_meta_level_output
            information types
              overall_design_strategy,
              design_process_evaluations;
          end information type

    private kernel information
      initial kernel information
      level level_2
        target(main,
          includes_design_strategy(S: design_process_state, N: design_strategy_name), confirm);
end component /* DPC */
```

```
component RQSM
  task information
    public task information
      task control foci
        main;
      evaluation criteria
        main;
    private task information
      components
        RQSM_history_maintenance, RQS_modification,
        current_RQS_maintenance, deductive_RQS_refinement;
      information links
        overall_design_strategy_for_RQSM,
        RQS_for_RQSM,
        DOD_assessments_for_RQSM,
        current_RQSM_trace_information,
        current_RQS_modification_basis,
        current_RQSM_step_information,
        current_RQS_modification_results,
        current_RQS_contents_assumptions_to_be_used,
        current_RQS_contents_information_to_be_used,
        current_design_requirement_information,
        current_deductive_RQS_refinement_focus,
        refined_design_requirement_information,
        process_evaluations_from_RQSM,
        RQS_from_RQSM,
        RQS_assessments_from_RQSM;
      initial task information
        task control focus main;
        extent all_p;
      task control contents
    knowledge base RQSM_task_control
      information types RQSM_task_control_sig
      contents
        if start
        then next_component_state(RQSM_history_maintenance, active)
         and next_task_control_focus(RQSM_history_maintenance, commencement)
         and next_link_state(overall_design_strategy_for_RQSM, uptodate)
         and next_link_state(RQS_for_RQSM, uptodate)
         and next_link_state(DOD_assessments_for_RQSM, uptodate);
```

if previous_component_state(RQSM_history_maintenance, active)
 and component_state(RQSM_history_maintenance, idle)
 and task_control_focus(RQSM_history_maintenance, commencement)
then next_component_state(RQS_modification, active)
 and next_link_state(current_RQSM_trace_information, uptodate)
 and next_link_state(current_RQS_modification_basis, uptodate);

if previous_component_state(RQS_modification, active)
 and component_state(RQS_modification, idle)
then next_link_state(current_RQSM_step_information, uptodate)
 and next_link_state(current_RQS_modification_results, uptodate);

if previous_component_state(RQS_modification, active)
 and component_state(RQS_modification, idle)
 and evaluation(RQS_modification, termination, any, succeeded)
then next_component_state(RQSM_history_maintenance, active)
 and next_task_control_focus(RQSM_history_maintenance, termination);

if previous_component_state(RQSM_history_maintenance, active)
 and task_control_focus(RQSM_history_maintenance, termination)
 and component_state(RQSM_history_maintenance, idle)
then stop
 and next_link_state(process_evaluations_from_RQSM, uptodate)
 and next_link_state(RQS_from_RQSM, uptodate)
 and next_link_state(RQS_assessments_from_RQSM, uptodate);

if previous_component_state(RQS_modification, active)
 and component_state(RQS_modification, idle)
 and evaluation(RQS_modification, query_and_retrieval, any, succeeded)
then next_component_state(RQSM_history_maintenance, active)
 and next_task_control_focus(RQSM_history_maintenance, processing);

if previous_component_state(RQSM_history_maintenance, active)
 and task_control_focus(RQSM_history_maintenance, processing)
 and component_state(RQSM_history_maintenance, idle)
then next_component_state(RQS_modification, active)
 and next_link_state(current_RQSM_trace_information, uptodate)
 and next_link_state(current_RQS_modification_basis, uptodate);

if previous_component_state(RQS_modification, active)
 and component_state(RQS_modification, idle)
 and evaluation(RQS_modification, replacement, any, succeeded)
then next_component_state(RQSM_history_maintenance, active)
 and next_task_control_focus(RQSM_history_maintenance, replacement);

if previous_component_state(RQSM_history_maintenance, active)
 and task_control_focus(RQSM_history_maintenance, replacement)
 and component_state(RQSM_history_maintenance, idle)
then next_component_state(current_RQS_maintenance, active)
 and next_link_state(current_RQS_contents_assumptions_to_be_used, uptodate);

if previous_component_state(RQS_modification, active)
 and component_state(RQS_modification, idle)
 and evaluation(RQS_modification, modification, any, succeeded)
then next_component_state(RQSM_history_maintenance, active)
 and next_task_control_focus(RQSM_history_maintenance, modification);

if previous_component_state(RQSM_history_maintenance, active)
 and task_control_focus(RQSM_history_maintenance, modification)
 and component_state(RQSM_history_maintenance, idle)
then next_component_state(current_RQS_maintenance, active)
 and next_link_state(current_RQS_contents_assumptions_to_be_used, uptodate);

if previous_component_state(current_RQS_maintenance, active)
 and component_state(current_RQS_maintenance, idle)
 and evaluation(RQS_modification, replacement, any, succeeded)
then next_component_state(RQS_modification, active)
 and next_link_state(current_RQSM_trace_information, uptodate)
 and next_link_state(current_RQS_modification_basis, uptodate);

if previous_component_state(current_RQS_maintenance, active)
 and component_state(current_RQS_maintenance, idle)
 and evaluation(RQS_modification, modification, any, succeeded)
then next_component_state(RQS_modification, active)
 and next_link_state(current_RQSM_trace_information, uptodate)
 and next_link_state(current_RQS_modification_basis, uptodate);

if previous_component_state(RQS_modification, active)
 and component_state(RQS_modification, idle)
 and evaluation(RQS_modification, deductive_refinement, any, succeeded)

```
        then next_component_state(deductive_RQS_refinement, active)
         and next_link_state(current_deductive_RQS_refinement_focus, uptodate)
         and next_link_state(current_design_requirement_information, uptodate);

        if previous_component_state(deductive_RQS_refinement, active)
         and component_state(deductive_RQS_refinement, idle)
        then next_component_state(current_RQS_maintenance, active)
         and next_link_state(refined_design_requirement_information, uptodate);

        if previous_component_state(current_RQS_maintenance, active)
         and component_state(current_RQS_maintenance, idle)
         and evaluation(RQS_modification, deductive_refinement, any, succeeded)
        then next_component_state(RQSM_history_maintenance, active)
         and task_control_focus(RQSM_history_maintenance, processing)
         and next_link_state(current_RQS_contents_information_to_be_used, uptodate);
    end knowledge base /* RQSM_task_control */

kernel information
  public kernel information
    knowledge structures
      information type current_RQS_contents_information
        information types
          design_requirement_info_element_type,
          sign_type;
        relations
          is_currently_included: design_requirement_info_element * sign;
      end information type

      information type current_RQS_contents_assumptions
        information types
          design_requirement_info_element_type,
          sign_type;
        relations
          is_to_be_asserted, is_to_be_retracted: design_requirement_info_element * sign;
      end information type

      information type RQS_modification_type
        information types
          design_requirement_info_element_type;
        sorts
          RQS_modification
```

```
  functions
    addition_of, deletion_of: design_requirement_info_element -> RQS_modification;
end information type

information type RQS_alteration_type
  information types
    RQS_modification_type;
  sorts
    RQS_alteration
  subsorts
    RQS_modification: RQS_alteration;
  objects
    NoAlteration: RQS_alteration;
end information type

information type RQS_alteration_composition_information
  information types
    RQS_alteration_type;
  relations
    includes_RQS_modification: RQS_alteration * RQS_modification;
end information type

information type proposed_RQS_alterations
  information types
    RQS_alteration_type;
  relations
    is_proposed_RQS_alteration: RQS_alteration;
end information type

information type rejected_RQS_alterations
  information types
    RQS_alteration_type;
  relations
    is_rejected_RQS_alteration: RQS_alteration;
end information type

information type application_specific_RQS_alteration_information
  information types
    RQS_name_type,
    RQS_alteration_type;
end information type
```

```
information type RQS_modification_basis
  information types
    RQS,
    DOD_assessments,
    RQS_assessments,
    current_RQS_identity_information,
    proposed_RQS_alterations,
    rejected_RQS_alterations,
    RQS_alteration_composition_information,
    application_specific_RQS_alteration_information,
    RQS_solution_information;
end information type

information type current_RQS_replacement_information
  information types
    RQS_name_type;
  relations
    is_replacement_for_current_RQS: RQS_name;
end information type

information type selected_RQS_alterations
  information types
    RQS_alteration_type;
  relations
    is_selected_RQS_alteration: RQS_alteration;
end information type

information type deductive_RQS_refinement_focus
  information types
    design_requirement_info_element_type,
    sign_type;
  relations
    is_part_of_deductive_RQS_refinement_focus:
     design_requirement_info_element * sign;
end information type

information type RQS_contents_information_queries
  information types
    RQS_name_type,
    design_requirement_info_element_type,
    sign_type;
```

```
  relations
    includes_which_design_requirement_information: RQS_name;
    is_included_in_which_RQSs: design_requirement_info_element * sign;
end information type

information type RQS_modification_results
  information types
    RQS_assessments,
    current_RQS_replacement_information,
    proposed_RQS_alterations,
    selected_RQS_alterations,
    RQS_alteration_composition_information,
    deductive_RQS_refinement_focus,
    RQS_contents_information_queries,
    application_specific_RQS_alteration_information,
    RQS_solution_information;
end information type

information type RQS_modification_process_information
  information types
    RQS_modification_basis,
    RQS_modification_results;
end information type

information type RQS_modification_process_info_element_type
  sorts
    RQS_modification_process_info_element
  meta-descriptions
    RQS_modification_process_information : RQS_modification_process_info_element;
end information type

information type state_specific_RQS_modification_process_information
  information types
    design_process_state_type, RQS_modification_process_info_element_type,
    sign_type;
  relations
    includes_RQS_modification_process_information:
     design_process_state * RQS_modification_process_info_element * sign;
end information type
```

```
information type application_specific_RQSM_process_state_information
  information types
    design_process_state_type;
end information type

information type RQSM_trace_information
  information types
    state_specific_RQS_modification_process_information,
    application_specific_RQSM_process_state_information,
    manipulation_trace_information;
end information type

information type state_specific_RQS_modification_process_information_queries
  information types
    design_process_state_type,
    RQS_modification_process_info_element_type,
    sign_type;
  relations
    includes_which_RQS_modification_process_information: design_process_state;
    is_included_in_which_design_process_states:
      RQS_modification_process_info_element * sign;
end information type

information type application_specific_RQSM_process_state_information_queries
  information types
    design_process_state_type;
end information type

information type RQSM_step_information
  information types
    current_control_decision_information,
    RQSM_process_evaluations,
    state_specific_RQS_modification_process_information_queries,
    application_specific_RQSM_process_state_information_queries,
    manipulation_trace_information_queries;
end information type

public levels
  level_1 , level_2;
public level chain
  level_1 < level_2;
```

```
        input interface
          level level_1
            information type RQSM_2nd_meta_level_input
              information types
                RQS,
                DOD_assessments;
            end information type
          level level_2
            information type overall_design_strategy;

        output interface
          level level_1
            information type RQSM_2nd_meta_level_output
              information types
                RQS,
                RQS_assessments;
            end information type
          level level_2
            information type RQSM_process_evaluations;

      private kernel information
        initial kernel information
        level level_3
          target(main, has_RQSM_process_evaluation(
            N: design_strategy_name, E: control_process_evaluation), confirm);

component RQSM_history_maintenance
  task information
    public task information
      task control foci
        commencement,
        processing,
        replacement,
        modification,
        termination;
    private task information
      initial task information
        extent all_p;
      task control contents
        knowledge bases
          empty_task_control_kbs;
```

```
kernel information
  public kernel information
    knowledge structures
    public levels
      level_1 , level_2;
    public level chain
      level_1 < level_2;

    input interface
      level level_1
        information type RQSM_history_maintenance_2nd_meta_level_input
          information types
            RQS,
            DOD_assessments,
            RQS_modification_results,
            current_RQS_contents_information;
        end information type
      level level_2
        information type RQSM_history_maintenance_3rd_meta_level_input
          information types
            overall_design_strategy,
            RQSM_step_information;
        end information type

    output interface
      level level_1
        information type RQSM_history_maintenance_2nd_meta_level_output
          information types
            RQS_modification_basis,
            current_RQS_contents_assumptions;
        end information type
      level level_2
        information type RQSM_trace_information;

  private kernel information
end component /* RQSM_history_maintenance */
```

```
component RQS_modification
  task information
    public task information
      task control foci
        main,
        termination,
        replacement,
        modification,
        deductive_refinement,
        query_and_retrieval;
      evaluation criteria
        termination,
        replacement,
        modification,
        deductive_refinement,
        query_and_retrieval;
    private task information
      initial task information
        task control focus main;
        extent all_p;
      task control contents
        knowledge bases
          empty_task_control_kbs;

  kernel information
    public kernel information
      knowledge structures
      public levels
        level_1 , level_2;
      public level chain
        level_1 < level_2;
      input interface
        level level_1
          information type RQS_modification_basis;
        level level_2
          information type RQSM_trace_information;
      output interface
        level level_1
          information type RQS_modification_results;
        level level_2
          information type RQSM_step_information;
```

```
    private kernel information
      initial kernel information
      level level_3
        target(main, is_current_control_decision(D: control_decision), confirm);
        target(termination, is_current_control_decision(termination), confirm);
        target(replacement, is_current_control_decision(replacement), confirm);
        target(modification, is_current_control_decision(modification), confirm);
        target(deductive_refinement, is_current_control_decision(deductive_refinement), confirm);
        target(query_and_retrieval, is_current_control_decision(query_and_retrieval), confirm);
end component /* RQS_modification */

component current_RQS_maintenance
  task information
    public task information
      task control foci
        main;
      evaluation criteria
        main;
    private task information
      initial task information
        task control focus main;
        extent all_p;
      task control contents
        standard

  kernel information
    public kernel information
      public levels
        level_1 , level_2;
      public level chain
        level_1 < level_2;

      input interface
        level level_1
          information type design_requirement_information;

      output interface
        level level_1
          information type design_requirement_information;
```

```
    private kernel information
      initial kernel information
      level level_2
        target(main, X: OA, determine);
      knowledge bases
        empty_primitive_knowledge_base
end component /* current_RQS_maintenance */

component deductive_RQS_refinement
  task information
    public task information
      task control foci
        main;
      evaluation criteria
        main;
    private task information
      initial task information
        task control focus main;
        extent all_p;
      task control contents
        standard

  kernel information
    public kernel information
      public levels
        level_1 , level_2;
      public level chain
        level_1 < level_2;
      input interface
        level level_1
          information type design_requirement_information;
      output interface
        level level_1
          information type design_requirement_information;
    private kernel information
      knowledge bases
        empty_primitive_knowledge_base
end component /* deductive_RQS_refinement */
```

```
private link current_RQSM_step_information: object - object
  domain RQS_modification
    level level_2
      information types RQSM_step_information;
  co-domain RQSM_history_maintenance
    level level_2
      information types RQSM_step_information;
  identity
end link /* current_RQSM_step_information */

private link current_design_requirement_information: object - object
  domain current_RQS_maintenance
    level level_1
      information types design_requirement_information;
  co-domain deductive_RQS_refinement
    level level_1
      information types design_requirement_information;
  identity
end link /* current_design_requirement_information */

private link current_deductive_RQS_refinement_focus: object - target
  domain RQS_modification
    level level_1
      information types deductive_RQS_refinement_focus;
  co-domain deductive_RQS_refinement
    level level_2
  sort links
   (design_requirement_info_element, OA)
   rest identity
  object links identity
  term links identity
  atom links
    (is_part_of_deductive_RQS_refinement_focus(E: design_requirement_info_element, pos),
     target(main, E: OA, confirm)):
      <<true, true>, <false, false>, <unknown, false>>;
    (is_part_of_deductive_RQS_refinement_focus(E: design_requirement_info_element, neg),
     target(main, E: OA, reject)):
      <<true, true>, <false, false>, <unknown, false>>;
end link /* current_deductive_RQS_refinement_focus */
```

```
private link current_RQSM_trace_information: object - object
  domain RQSM_history_maintenance
    level level_2
      information types RQSM_trace_information;
  co-domain RQS_modification
    level level_2
      information types RQSM_trace_information;
  identity
end link /* current_RQSM_trace_information */

private link current_RQS_modification_basis: object - object
  domain RQSM_history_maintenance
    level level_1
      information types RQS_modification_basis;
  co-domain RQS_modification
    level level_1
      information types RQS_modification_basis;
  identity
end link /* current_RQS_modification_basis */

private link current_RQS_contents_assumptions_to_be_used: object - assumption
  domain RQSM_history_maintenance
    level level_1
      information types current_RQS_contents_assumptions;
  co-domain current_RQS_maintenance
    level level_2
  sort links
   (design_requirement_info_element, IA)
   (sign, Signs)
   rest identity
  object links identity
  term links identity
  atom links
    (is_to_be_asserted(E: design_requirement_info_element, S: sign),
     assumption(E: IA, S: Signs)):
      <<true, true>>;
    (is_to_be_retracted(E: design_requirement_info_element, S: sign),
     assumption(E: IA, S: Signs)):
      <<true, false>>;
end link /* current_RQS_contents_assumptions_to_be_used */
```

```
private link current_RQS_contents_information_to_be_used: epistemic - object
  domain current_RQS_maintenance
    level level_2
  co-domain RQSM_history_maintenance
    level level_1
      information types current_RQS_contents_information;
  sort links
   (OA, design_requirement_info_element)
   rest identity
  object links identity
  term links identity
  atom links
   (true(A: OA), is_currently_included(A: design_requirement_info_element, pos)):
     <<true, true>, <false, unknown>>;
   (false(A: OA), is_currently_included(A: design_requirement_info_element, neg)):
     <<true, true>, <false, unknown>>;
end link /* current_RQS_contents_information_to_be_used */

private link refined_design_requirement_information: object - object
  domain deductive_RQS_refinement
    level level_1
      information types design_requirement_information;
  co-domain current_RQS_maintenance
    level level_1
      information types design_requirement_information;
  identity
end link /* refined_design_requirement_information */

private link current_RQS_modification_results: object - object
  domain RQS_modification
    level level_1
      information types RQS_modification_results;
  co-domain RQSM_history_maintenance
    level level_1
      information types RQS_modification_results;
  identity
end link /* current_RQS_modification_results */
```

```
mediating link RQS_for_RQSM: object - object
  domain RQSM
    level level_1
      information types RQS;
  co-domain RQSM_history_maintenance
    level level_1
      information types RQS;
  identity
end link /* RQS_for_RQSM */

mediating link DOD_assessments_for_RQSM: object - object
  domain RQSM
    level level_1
      information types DOD_assessments;
  co-domain RQSM_history_maintenance
    level level_1
      information types DOD_assessments;
  identity
end link /* DOD_assessments_for_RQSM */

mediating link RQS_from_RQSM: object - object
  domain RQSM_history_maintenance
    level level_1
      information types RQS;
  co-domain RQSM
    level level_1
      information types RQS;
  identity
end link /* RQS_from_RQSM */

mediating link RQS_assessments_from_RQSM: object - object
  domain RQSM_history_maintenance
    level level_1
      information types RQS_assessments;
  co-domain RQSM
    level level_1
      information types RQS_assessments;
  identity
end link /* RQS_assessments_from_RQSM */
```

```
mediating link overall_design_strategy_for_RQSM: object - object
  domain RQSM
    level level_2
      information types overall_design_strategy;
  co-domain RQSM_history_maintenance
    level level_2
      information types overall_design_strategy;
  identity
end link /* overall_design_strategy_for_RQSM */

mediating link process_evaluations_from_RQSM: object - object
  domain RQS_modification
    level level_2
      information types RQSM_process_evaluations;
  co-domain RQSM
    level level_2
      information types RQSM_process_evaluations;
  identity
end link /* process_evaluations_from_RQSM */
end component /* RQSM */

component DODM
  task information
    public task information
      task control foci
        main;
      evaluation criteria
        main;
    private task information
      components
        DODM_history_maintenance,
        DOD_modification,
        current_DOD_maintenance,
        deductive_DOD_refinement;
      information links
        overall_design_strategy_for_DODM,
        RQS_for_DODM,
        DOD_for_DODM,
        current_DODM_trace_information,
        current_DOD_modification_basis,
        current_DODM_step_information,
```

```
    current_DOD_modification_results,
    current_DOD_contents_information_to_be_used,
    current_DOD_contents_assumptions_to_be_used,
    current_domain_object_information,
    current_deductive_DOD_refinement_focus,
    refined_domain_object_information,
    process_evaluations_from_DODM,
    DOD_from_DODM,
    DOD_assessments_from_DODM;
  initial task information
    task control focus main;
    extent all_p;
  task control contents
knowledge base DODM_task_control
  information types DODM_task_control_sig
  contents
    if start
    then next_component_state(DODM_history_maintenance, active)
     and next_task_control_focus(DODM_history_maintenance, commencement)
     and next_link_state(overall_design_strategy_for_DODM, uptodate)
     and next_link_state(RQS_for_DODM, uptodate)
     and next_link_state(DOD_for_DODM, uptodate);

    if previous_component_state(DODM_history_maintenance, active)
     and component_state(DODM_history_maintenance, idle)
     and task_control_focus(DODM_history_maintenance, commencement)
    then next_component_state(DOD_modification, active)
     and next_link_state(current_DODM_trace_information, uptodate)
     and next_link_state(current_DOD_modification_basis, uptodate);

    if previous_component_state(DOD_modification, active)
     and component_state(DOD_modification, idle)
    then next_link_state(current_DODM_step_information, uptodate)
     and next_link_state(current_DOD_modification_results, uptodate);

    if previous_component_state(DOD_modification, active)
     and component_state(DOD_modification, idle)
     and evaluation(DOD_modification, termination, any, succeeded)
    then next_component_state(DODM_history_maintenance, active)
     and next_task_control_focus(DODM_history_maintenance, termination);
```

if previous_component_state(DODM_history_maintenance, active)
 and task_control_focus(DODM_history_maintenance, termination)
 and component_state(DODM_history_maintenance, idle)
then stop
 and next_link_state(process_evaluations_from_DODM, uptodate)
 and next_link_state(DOD_from_DODM, uptodate)
 and next_link_state(DOD_assessments_from_DODM, uptodate);

if previous_component_state(DOD_modification, active)
 and component_state(DOD_modification, idle)
 and evaluation(DOD_modification, query_and_retrieval, any, succeeded)
then next_component_state(DODM_history_maintenance, active)
 and next_task_control_focus(DODM_history_maintenance, processing);

if previous_component_state(DODM_history_maintenance, active)
 and task_control_focus(DODM_history_maintenance, processing)
 and component_state(DODM_history_maintenance, idle)
then next_component_state(DOD_modification, active)
 and next_link_state(current_DODM_trace_information, uptodate)
 and next_link_state(current_DOD_modification_basis, uptodate);

if previous_component_state(DOD_modification, active)
 and component_state(DOD_modification, idle)
 and evaluation(DOD_modification, replacement, any, succeeded)
then next_component_state(DODM_history_maintenance, active)
 and next_task_control_focus(DODM_history_maintenance, replacement);

if previous_component_state(DODM_history_maintenance, active)
 and task_control_focus(DODM_history_maintenance, replacement)
 and component_state(DODM_history_maintenance, idle)
then next_component_state(current_DOD_maintenance, active)
 and next_link_state(current_DOD_contents_assumptions_to_be_used, uptodate);

if previous_component_state(DOD_modification, active)
 and component_state(DOD_modification, idle)
 and evaluation(DOD_modification, modification, any, succeeded)
then next_component_state(DODM_history_maintenance, active)
 and next_task_control_focus(DODM_history_maintenance, modification);

```
    if previous_component_state(DODM_history_maintenance, active)
     and task_control_focus(DODM_history_maintenance, modification)
     and component_state(DODM_history_maintenance, idle)
    then next_component_state(current_DOD_maintenance, active)
     and next_link_state(current_DOD_contents_assumptions_to_be_used, uptodate);

    if previous_component_state(current_DOD_maintenance, active)
     and component_state(current_DOD_maintenance, idle)
     and evaluation(DOD_modification, replacement, any, succeeded)
    then next_component_state(DOD_modification, active)
     and next_link_state(current_DODM_trace_information, uptodate)
     and next_link_state(current_DOD_modification_basis, uptodate);

    if previous_component_state(current_DOD_maintenance, active)
     and component_state(current_DOD_maintenance, idle)
     and evaluation(DOD_modification, modification, any, succeeded)
    then next_component_state(DOD_modification, active)
     and next_link_state(current_DODM_trace_information, uptodate)
     and next_link_state(current_DOD_modification_basis, uptodate);

    if previous_component_state(DOD_modification, active)
     and component_state(DOD_modification, idle)
     and evaluation(DOD_modification, deductive_refinement, any, succeeded)
    then next_component_state(deductive_DOD_refinement, active)
     and next_link_state(current_deductive_DOD_refinement_focus, uptodate)
     and next_link_state(current_domain_object_information, uptodate);

    if previous_component_state(deductive_DOD_refinement, active)
     and component_state(deductive_DOD_refinement, idle)
    then next_component_state(current_DOD_maintenance, active)
     and next_link_state(refined_domain_object_information, uptodate);

    if previous_component_state(current_DOD_maintenance, active)
     and component_state(current_DOD_maintenance, idle)
     and evaluation(DOD_modification, deductive_refinement, any, succeeded)
    then next_component_state(DODM_history_maintenance, active)
     and next_task_control_focus(DODM_history_maintenance, processing)
     and next_link_state(current_DOD_contents_information_to_be_used, uptodate);
end knowledge base /* DODM_task_control */
```

```
kernel information
  public kernel information
    knowledge structures
      information type current_DOD_contents_information
        information types
          domain_object_info_element_type,
          sign_type;
        relations
          is_currently_included: domain_object_info_element * sign;
      end information type

      information type current_DOD_contents_assumptions
        information types
          domain_object_info_element_type,
          sign_type;
        relations
          is_to_be_asserted, is_to_be_retracted: domain_object_info_element * sign;
      end information type

      information type default_domain_object_information
        information types
          DOD_name_type, domain_object_info_element_type, sign_type;
        relations
          has_default_domain_object_information:
            DOD_name * domain_object_info_element * sign;
      end information type

      information type application_specific_DOD_information
        information types
          DOD_name_type, domain_object_info_element_type, sign_type;
      end information type

      information type DOD_specific_basis_information
        information types
          DOD,
          default_domain_object_information,
          application_specific_DOD_information,
          design_requirement_information,
          basic_DOD_assessments;
      end information type
```

```
information type deductive_DOD_refinement_focus
  information types
    domain_object_info_element_type,
    sign_type;
  relations
    is_part_of_deductive_DOD_refinement_focus: domain_object_info_element * sign;
end information type

information type DOD_contents_information_queries
  information types
    DOD_name_type,
    domain_object_info_element_type,
    sign_type;
  relations
    includes_which_domain_object_information: DOD_name;
    is_included_in_which_DODs: domain_object_info_element * sign;
end information type

information type DOD_specific_results_information
  information types
    default_domain_object_information,
    application_specific_DOD_information,
    deductive_DOD_refinement_focus,
    DOD_contents_information_queries,
    basic_DOD_assessments;
end information type

information type DOD_modification_type
  information types
    domain_object_info_element_type,
    sign_type;
  sorts
    domain_object_information_literal,
    DOD_modification
  functions
    domain_object_information:
      domain_object_info_element * sign -> domain_object_information_literal;
    addition_of, deletion_of: domain_object_information_atom -> DOD_modification;
end information type
```

```
information type DOD_alteration_type
  information types
    DOD_modification_type;
  sorts
    DOD_alteration
  subsorts
    DOD_modification: DOD_alteration;
  objects
    NoAlteration: DOD_alteration;
end information type

information type DOD_alteration_composition_information
  information types
    DOD_alteration_type;
  relations
    includes_DOD_modification: DOD_alteration * DOD_modification;
end information type

information type proposed_DOD_alterations
  information types
    DOD_alteration_type;
  relations
    is_proposed_DOD_alteration: DOD_alteration;
end information type

information type rejected_DOD_alterations
  information types
    DOD_alteration_type;
  relations
    is_rejected_DOD_alteration: DOD_alteration;
end information type

information type application_specific_DOD_alteration_information
  information types
    DOD_name_type,
    DOD_alteration_type;
end information type

information type DOD_specific_basis_info_element_type
  sorts
    DOD_specific_basis_info_element
```

```
  meta-descriptions
    DOD_specific_basis_information : DOD_specific_basis_info_element;
end information type

information type epistemic_DOD_specific_basis_information
  information types
    DOD_specific_basis_info_element_type,
    sign_type;
  relations
    is_part_of_DOD_modification_basis: DOD_specific_basis_info_element * sign;
end information type

information type DOD_modification_basis
  information types
    epistemic_DOD_specific_basis_information,
    RQS,
    DOD_assessments,
    current_RQS_identity_information,
    current_DOD_identity_information,
    proposed_DOD_alterations,
    rejected_DOD_alterations,
    DOD_alteration_composition_information,
    application_specific_DOD_alteration_information,
    DOD_solution_information;
end information type

information type current_DOD_replacement_information
  information types
    DOD_name_type;
  relations
    is_replacement_for_current_DOD: DOD_name;
end information type

information type selected_DOD_alterations
  information types
    DOD_alteration_type;
  relations
    is_selected_DOD_alteration: DOD_alteration;
end information type
```

information type DOD_specific_results_info_element_type
  sorts
    DOD_specific_results_info_element
  meta-descriptions
    DOD_specific_results_information : DOD_specific_results_info_element;
end information type

information type epistemic_DOD_specific_results_information
  information types
    DOD_specific_results_info_element_type,
    sign_type;
  relations
    is_part_of_DOD_modification_results: DOD_specific_results_info_element * sign;
end information type

information type DOD_modification_results
  information types
    epistemic_DOD_specific_results_information,
    DOD_assessments,
    current_DOD_replacement_information,
    proposed_DOD_alterations,
    selected_DOD_alterations,
    DOD_alteration_composition_information,
    application_specific_DOD_alteration_information,
    DOD_solution_information;
end information type

information type DOD_modification_process_information
  information types
    DOD_modification_basis,
    DOD_modification_results;
end information type

information type DOD_modification_process_info_element_type
  sorts
    DOD_modification_process_info_element
  meta-descriptions
    DOD_modification_process_information : DOD_modification_process_info_element;
end information type

```
information type state_specific_DOD_modification_process_information
  information types
    design_process_state_type, DOD_modification_process_info_element_type,
    sign_type;
  relations
    includes_DOD_modification_process_information:
     design_process_state * DOD_modification_process_info_element * sign;
end information type

information type application_specific_DODM_process_state_information
  information types
    design_process_state_type;
end information type

information type DODM_trace_information
  information types
    state_specific_DOD_modification_process_information,
    application_specific_DODM_process_state_information,
    manipulation_trace_information;
end information type

information type state_specific_DOD_modification_process_information_queries
  information types
    design_process_state_type,
    DOD_modification_process_info_element_type,
    sign_type;
  relations
    includes_which_DOD_modification_process_information: design_process_state;
    is_included_in_which_design_process_states:
      DOD_modification_process_info_element * sign;
end information type

information type application_specific_DODM_process_state_information_queries
  information types
    design_process_state_type;
end information type

information type DODM_step_information
  information types
    current_control_decision_information,
    DODM_process_evaluations,
```

```
            state_specific_DOD_modification_process_information_queries,
            application_specific_DODM_process_state_information_queries,
            manipulation_trace_information_queries;
        end information type

      public levels
        level_1 , level_2 , level_3;
      public level chain
        level_1 < level_2 < level_3;
      input interface
        level level_1
          information type DOD;
        level level_2
          information type RQS;
        level level_3
          information type overall_design_strategy;
      output interface
        level level_1
          information type DOD;
        level level_2
          information type DOD_assessments;
        level level_3
          information type DODM_process_evaluations;
    private kernel information
      initial kernel information
      level level_4
        target(main, has_DODM_process_evaluation(
          N: design_strategy_name, E: control_process_evaluation), confirm);

component DODM_history_maintenance
  task information
    public task information
      task control foci
        commencement,
        processing,
        replacement,
        modification,
        termination;
    private task information
      initial task information
        extent all_p;
```

```
    task control contents
      knowledge bases
        empty_task_control_kbs;

kernel information
  public kernel information
    knowledge structures
    public levels
      level_1 , level_2 , level_3;
    public level chain
      level_1 < level_2 < level_3;

    input interface
      level level_1
        information type DODM_history_maintenance_1st_meta_level_input
          information types
            DOD,
            current_DOD_contents_information,
            DOD_specific_results_information;
        end information type
      level level_2
        information type DODM_history_maintenance_2nd_meta_level_input
          information types
            RQS,
            DOD_modification_results;
        end information type
      level level_3
        information type DODM_history_maintenance_3rd_meta_level_input
          information types
            overall_design_strategy,
            DODM_step_information;
        end information type

    output interface
      level level_1
        information type DODM_history_maintenance_1st_meta_level_output
          information types
            DOD,
            current_DOD_contents_assumptions,
            DOD_specific_basis_information;
        end information type
```

```
        level level_2
          information type DOD_modification_basis;
        level level_3
          information type DODM_trace_information;

    private kernel information
end component /* DODM_history_maintenance */

component DOD_modification
  task information
    public task information
      task control foci
        main,
        termination,
        replacement,
        modification,
        deductive_refinement,
        query_and_retrieval;
      evaluation criteria
        termination,
        replacement,
        modification,
        deductive_refinement,
        query_and_retrieval;
    private task information
      initial task information
        task control focus main;
        extent all_p;
      task control contents
        knowledge bases
          empty_task_control_kbs;

  kernel information
    public kernel information
      knowledge structures
      public levels
        level_1 , level_2 , level_3;
      public level chain
        level_1 < level_2 < level_3;
```

```
        input interface
          level level_1
            information type DOD_specific_basis_information;
          level level_2
            information type DOD_modification_basis;
          level level_3
            information type DODM_trace_information;

        output interface
          level level_1
            information type DOD_specific_results_information;
          level level_2
            information type DOD_modification_results;
          level level_3
            information type DODM_step_information;

      private kernel information
        initial kernel information
        level level_4
          target(main, is_current_control_decision(D: control_decision), confirm);
          target(termination, is_current_control_decision(termination), confirm);
          target(replacement, is_current_control_decision(replacement), confirm);
          target(modification, is_current_control_decision(modification), confirm);
          target(deductive_refinement, is_current_control_decision(deductive_refinement), confirm);
          target(query_and_retrieval, is_current_control_decision(query_and_retrieval), confirm);
end component /* DOD_modification */

component current_DOD_maintenance
  task information
    public task information
      task control foci
        main;
      evaluation criteria
        main;
    private task information
      initial task information
        task control focus main;
        extent all_p;
      task control contents
        standard
```

```
  kernel information
    public kernel information
      public levels
        level_1 , level_2;
      public level chain
        level_1 < level_2;
      input interface
        level level_1
          information type domain_object_information;
      output interface
        level level_1
          information type domain_object_information;
    private kernel information
      initial kernel information
      level level_2
        target(main, X: OA, determine);
      knowledge bases
        empty_primitive_knowledge_base
end component /* current_DOD_maintenance */

component deductive_DOD_refinement
  task information
    public task information
      task control foci
        main;
      evaluation criteria
        main;
    private task information
      initial task information
        extent all_p;
      task control contents
        standard

  kernel information
    public kernel information
      public levels
        level_1 , level_2;
      public level chain
        level_1 < level_2;
```

```
        input interface
          level level_1
            information type domain_object_information;
        output interface
          level level_1
            information type domain_object_information;

      private kernel information
        knowledge bases
          empty_primitive_knowledge_base
end component /* deductive_DOD_refinement */

private link current_DOD_modification_basis: object - object
  domain DODM_history_maintenance
    level level_2
      information types DOD_modification_basis;
  co-domain DOD_modification
    level level_2
      information types DOD_modification_basis;
  identity
end link /* current_DOD_modification_basis */

private link current_domain_object_information: object - object
  domain current_DOD_maintenance
    level level_1
      information types domain_object_information;
  co-domain deductive_DOD_refinement
    level level_1
      information types domain_object_information;
  identity
end link /* current_domain_object_information */

private link refined_domain_object_information: object - object
  domain deductive_DOD_refinement
    level level_1
      information types domain_object_information;
  co-domain current_DOD_maintenance
    level level_1
      information types domain_object_information;
  identity
end link /* refined_domain_object_information */
```

```
private link current_deductive_DOD_refinement_focus: object - target
  domain DOD_modification
    level level_1
      information types deductive_DOD_refinement_focus;
  co-domain deductive_DOD_refinement
    level level_2
  sort links
   (domain_object_info_element, OA)
   rest identity
  object links identity
  term links identity
  atom links
   (is_part_of_deductive_DOD_refinement_focus(E: domain_object_info_element, pos),
    target(main, E: OA, confirm)):
     <<true, true>, <false, false>, <unknown, false>>;
   (is_part_of_deductive_DOD_refinement_focus(E: domain_object_info_element, neg),
    target(main, E: OA, reject)):
     <<true, true>, <false, false>, <unknown, false>>;
end link /* current_deductive_DOD_refinement_focus */

private link current_DOD_contents_information_to_be_used: epistemic - object
  domain current_DOD_maintenance
    level level_2
  co-domain DODM_history_maintenance
    level level_1
      information types current_DOD_contents_information;
  sort links
   (OA, domain_object_info_element)
   rest identity
  object links identity
  term links identity
  atom links
   (true(E: OA), is_currently_included(E: domain_object_info_element, pos)):
     <<true, true>, <false, unknown>>;
   (false(E: OA), is_currently_included(E: domain_object_info_element, neg)):
     <<true, true>, <false, unknown>>;
end link /* current_DOD_contents_information_to_be_used */
```

```
private link current_DOD_contents_assumptions_to_be_used: object - assumption
  domain DODM_history_maintenance
    level level_1
      information types current_DOD_contents_assumptions;
  co-domain current_DOD_maintenance
    level level_2
  sort links
   (domain_object_info_element, IA)
   (sign, Signs)
   rest identity
  object links identity
  term links identity
  atom links
   (is_to_be_asserted(E: domain_object_info_element, S: sign),
    assumption(E: IA, S: Signs)):
     <<true, true>>;
   (is_to_be_retracted(E: domain_object_info_element, S: sign),
    assumption(E: IA, S: Signs)):
     <<true, false>>;
end link /* current_DOD_contents_assumptions_to_be_used */

private link current_DOD_modification_results: object - object
  domain DOD_modification
    level level_2
      information types DOD_modification_results;
  co-domain DODM_history_maintenance
    level level_2
      information types DOD_modification_results;
  identity
end link /* current_DOD_modification_results */

mediating link DOD_for_DODM: object - object
  domain DODM
    level level_1
      information types DOD;
  co-domain DODM_history_maintenance
    level level_1
      information types DOD;
  identity
end link /* DOD_for_DODM */
```

```
mediating link DOD_from_DODM: object - object
  domain DODM_history_maintenance
    level level_1
      information types DOD;
  co-domain DODM
    level level_1
      information types DOD;
  identity
end link /* DOD_from_DODM */

mediating link overall_design_strategy_for_DODM: object - object
  domain DODM
    level level_3
      information types overall_design_strategy;
  co-domain DODM_history_maintenance
    level level_3
      information types overall_design_strategy;
  identity
end link /* overall_design_strategy_for_DODM */

mediating link process_evaluations_from_DODM: object - object
  domain DOD_modification
    level level_3
      information types DODM_process_evaluations;
  co-domain DODM
    level level_3
      information types DODM_process_evaluations;
  identity
end link /* process_evaluations_from_DODM */

mediating link DOD_assessments_from_DODM: object - object
  domain DODM_history_maintenance
    level level_2
      information types DOD_assessments;
  co-domain DODM
    level level_2
      information types DOD_assessments;
  identity
end link /* DOD_assessments_from_DODM */
```

```
mediating link RQS_for_DODM: object - object
  domain DODM
    level level_2
      information types RQS;
  co-domain DODM_history_maintenance
    level level_2
      information types RQS;
  identity
end link /* RQS_for_DODM */

private link current_DODM_trace_information: object - object
  domain DODM_history_maintenance
    level level_3
      information types DODM_trace_information;
  co-domain DOD_modification
    level level_3
      information types DODM_trace_information;
  identity
end link /* current_DODM_trace_information */

private link current_DODM_step_information: object - object
  domain DOD_modification
    level level_3
      information types DODM_step_information;
  co-domain DODM_history_maintenance
    level level_3
      information types DODM_step_information;
  identity
end link /* current_DODM_step_information */
end component /* DODM */

mediating link given_DOD: object - object
  domain design
    level level_1
      information types DOD;
  co-domain DODM
    level level_1
      information types DOD;
  identity
end link /* given_DOD */
```

```
mediating link resulting_DOD: object - object
  domain DODM
    level level_1
      information types DOD;
  co-domain design
    level level_1
      information types DOD;
  identity
end link /* resulting_DOD */

mediating link given_RQS: object - object
  domain design
    level level_2
      information types RQS;
  co-domain RQSM
    level level_1
      information types RQS;
  identity
end link /* given_RQS */

private link intermediate_RQS: object - object
  domain RQSM
    level level_1
      information types RQS;
  co-domain DODM
    level level_2
      information types RQS;
  identity
end link /* intermediate_RQS */

mediating link given_design_process_objectives: object - object
  domain design
    level level_3
      information types design_process_objectives;
  co-domain DPC
    level level_1
      information types design_process_objectives;
  identity
end link /* given_design_process_objectives */
```

```
mediating link resulting_DOD_assessments: object - object
  domain DODM
    level level_2
      information types DOD_assessments;
  co-domain design
    level level_2
      information types DOD_assessments;
  identity
end link /* resulting_DOD_assessments */

private link intermediate_RQSM_process_evaluations: object - object
  domain RQSM
    level level_2
      information types RQSM_process_evaluations;
  co-domain DPC
    level level_1
      information types RQSM_process_evaluations;
  identity
end link /* intermediate_RQSM_process_evaluations */

private link intermediate_DODM_process_evaluations: object - object
  domain DODM
    level level_3
      information types DODM_process_evaluations;
  co-domain DPC
    level level_1
      information types DODM_process_evaluations;
  identity
end link /* intermediate_DODM_process_evaluations */

mediating link resulting_design_process_evaluations: object - object
  domain DPC
    level level_1
      information types design_process_evaluations;
  co-domain design
    level level_3
      information types design_process_evaluations;
  identity
end link /* resulting_design_process_evaluations */
```

```
mediating link resulting_RQS: object - object
  domain RQSM
    level level_1
      information types RQS;
  co-domain design
    level level_2
      information types RQS;
  identity
end link /* resulting_RQS */

private link intermediate_DOD_assessments: object - object
  domain DODM
    level level_2
      information types DOD_assessments;
  co-domain RQSM
    level level_1
      information types DOD_assessments;
  identity
end link /* intermediate_DOD_assessments */

mediating link resulting_RQS_assessments: object - object
  domain RQSM
    level level_1
      information types RQS_assessments;
  co-domain design
    level level_2
      information types RQS_assessments;
  identity
end link /* resulting_RQS_assessments */

private link intermediate_overall_design_strategy_to_RQSM: object - object
  domain DPC
    level level_1
      information types overall_design_strategy;
  co-domain RQSM
    level level_2
      information types overall_design_strategy;
  identity
end link /* intermediate_overall_design_strategy_to_RQSM */
```

```
private link intermediate_overall_design_strategy_to_DODM: object - object
  domain DPC
    level level_1
      information types overall_design_strategy;
  co-domain DODM
    level level_3
      information types overall_design_strategy;
  identity
end link /* intermediate_overall_design_strategy_to_DODM */
end component /* design */
end component /* GDM */
```

Bibliography

AIHARA, K. & HORI, K. (1998). Enhancing creativity through reorganising mental space concealed in a research notes stack. In K. HORI, Ed., *Knowledge-Based Systems*, Special Issue on Strategic Knowledge and Concept Formation, **11**(7-8):469–478.

AKIN, Ö. (1978). How do architects design? In J.-C. LATOMBE, Ed., *Artificial Intelligence and Pattern Recognition in Computer Aided Design*, pp. 65–119. Amsterdam: North-Holland.

ALBERTS, L. K., WOGNUM, P. M. & MARS, N. J. I. (1992). Structuring design knowledge on the basis of generic components. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '92 (AID '92)*, pp. 639–656. Boston: Kluwer Academic Publishers.

ALEXANDER, C. (1964). *Notes on the Synthesis of Form*. Cambridge: Harvard University Press.

ALEXANDER, C. & POYNER, B. (1970). The atoms of environmental structure. In G. T. MOORE, Ed., *Emerging Methods in Environmental Design and Planning*, pp. 308–321. Cambridge: MIT Press.

ARCHER, L. B. (1970). An overview of the structure of the design process. In G. T. MOORE, Ed., *Emerging Methods in Environmental Design and Planning*, pp. 285–307. Cambridge: MIT Press.

ASIMOW, M. (1962). *Introduction to Design*. Englewood Cliffs: Prentice-Hall.

BERNARAS, A. & VAN DE VELDE, W. (1994). Design. In J. A. P. J. BREUKER & W. VAN DE VELDE, Eds., *CommonKADS Library for Expertise Modelling: Reusable Problem Solving Components*, pp. 175–196. Amsterdam: IOS Press.

BIJL, A. (1987). An approach to design theory. In H. YOSHIKAWA & E. A. WARMAN, Eds., *Proc. IFIP WG5.2 Working Conf. on Design Theory for CAD*, pp. 3–25. Amsterdam: Elsevier (North-Holland).

BLAMEY, S. (1986). Partial logic. In D. GABBAY & F. GUENTHNER, Eds., *Handbook of Philosophical Logic*, **III**:1–70. Dordrecht: Reidel.

BODEN, M. A. (1990). *The Creative Mind: Myths and Mechanisms*. London: Weidenfeld and Nicolson.

BRADLEY, S., AGOGINO, A. & WOOD, W. (1994). Intelligent engineering component catalogs. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '94*, pp. 641–658. Dordrecht: Kluwer Academic Publishers.

BRAZIER, F. M. T., CORNELISSEN, F., JONKER, C. M. & TREUR, J. (2000). Compositional specification and reuse of a generic cooperative agent model. *Int. J. Cooperative Information Systems*, **9**(3):171–207.

BRAZIER, F. M. T., JONKER, C. M. & TREUR, J. (1998). Principles of compositional multi-agent system development. In J. CUENA, Ed., *Proc. 15th IFIP World Computer Congress (WCC '98), Conf. Information Technology and Knowledge Systems (IT&KNOWS '98)*, pp. 347–360. Extended version (2002): Principles of component-based design of intelligent agents, *Data and Knowledge Engineering*, **41**:1–28.

BRAZIER, F. M. T., JONKER, C. M. & TREUR, J. (2000). Compositional design and reuse of a generic agent model. *Applied Artificial Intelligence*, **14**(5):491–538.

BRAZIER, F. M. T., JONKER, C. M., TREUR, J. & WIJNGAARDS, N. J. E. (2000). On the use of shared task models in knowledge acquisition, strategic user interaction and clarification agents. *Int. J. Human-Computer Studies*, **52**(1):77–110.

BRAZIER, F. M. T., KLERK, D. A. DE, LANGEN, P. H. G. VAN & TREUR, J. (1993). *A generic architecture for knowledge-based control of processes in dynamic environments*. Technical Report IR-347. Amsterdam: Vrije Universiteit, Faculty of Sciences, Department of Artificial Intelligence.

BRAZIER, F. M. T., LANGEN, P. H. G. VAN, RUTTKAY, ZS. & TREUR, J. (1994). On formal specification of design tasks. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '94 (AID '94)*, pp. 535–552. Dordrecht: Kluwer Academic Publishers.

BRAZIER, F. M. T., LANGEN, P. H. G. VAN & TREUR, J. (1995*a*). A logical theory of design. In J. S. GERO & F. SUDWEEKS, Eds., *Advances in Formal Design Methods for CAD*. Preprints IFIP WG5.2 Workshop on Formal Design Methods for CAD, pp. 247–271. Sydney: University of Sydney, Key Centre of Design Computing.

BRAZIER, F. M. T., LANGEN, P. H. G. VAN & TREUR, J. (1995*b*). Modelling conflict management in design: an explicit approach. In I. F. C. SMITH, Ed., *AIEDAM*, Special Issue on Conflict Management in Design, **9**(4):355–366.

BRAZIER, F. M. T., LANGEN, P. H. G. VAN & TREUR, J. (1996). A logical theory of design. In J. S. GERO, Ed., *Advances in Formal Design Methods for CAD*, pp. 243-266. New York: Chapman & Hall.

BRAZIER, F. M. T., LANGEN, P. H. G. VAN & TREUR, J. (1997). A compositional approach to modelling design rationale. In P. W. H. CHUNG & R. BAÑARES-ALCÁNTARA, Eds., *AIEDAM*, Special Issue on Representing and Using Design Rationale, **11**(2):125–139.

BRAZIER, F. M. T., LANGEN, P. H. G. VAN & TREUR, J. (1998*a*). Strategic knowledge in compositional design models. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '98 (AID '98)*, pp. 129–147. Dordrecht: Kluwer Academic Publishers.

BRAZIER, F. M. T., LANGEN, P. H. G. VAN & TREUR, J. (1998*b*). Strategic knowledge in design: a compositional approach. In K. HORI, Ed., *Knowledge-Based Systems*, Special Issue on Strategic Knowledge and Concept Formation, **11**(7-8):405–416.

BRAZIER, F. M. T., LANGEN, P. H. G. VAN, TREUR, J. & WIJNGAARDS, N. J. E. (1996). Redesign and reuse in compositional knowledge-based systems. In P. M. WOGNUM & I. F. C. SMITH, Eds., *Knowledge-Based Systems*, Special Issue on Models and Techniques for Reuse of Designs, **9**(2):105–118.

BRAZIER, F. M. T., LANGEN, P. H. G. VAN, TREUR, J., WIJNGAARDS, N. J. E. & WILLEMS, M. (1996). Modelling an elevator design task in DESIRE: the VT example. In A. TH. SCHREIBER & W. P. BIRMINGHAM, Eds., *Int. J. Human-Computer Studies*, Special Issue on Sisyphus-VT, **44**:469–520.

BRAZIER, F. M. T., TREUR, J. & WIJNGAARDS, N. J. E. (1996). Interaction with experts: the role of a shared task model. In W. WAHLSTER, Ed., *Proc. Europ. Conf. on AI (ECAI '96)*, pp. 241–245. Chichester: Wiley and Sons.

BREUKER, J. (1994). Components of problem solving and types of problems. In L. STEELS, A. TH. SCHREIBER & W. VAN DE VELDE, Eds., *A Future for Knowledge Acquisition*, Proc. Eighth Europ. Knowledge Acquisition Workshop (EKAW '94), Lecture Notes in Artificial Intelligence No. 867, pp. 128–136. Berlin: Springer-Verlag.

BROWN, D. C. & CHANDRASEKARAN, B. (1989). *Design Problem Solving: Knowledge Structures and Control Strategies*. Research Notes in Artificial Intelligence. London: Pitman.

BRUMSEN, H. A., PANNEKEET, J. H. M. & TREUR, J. (1992). A compositional knowledge-based architecture modelling process aspects of design tasks. *Proc. Twelfth Int. Conf. on Artificial Intelligence, Expert systems and Natural Language (Avignon-92)*, **1**:283–294. Nanterre: EC2.

BURGE, J. & BROWN, D. C. (2000). Reasoning with design rationale. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '00 (AID '00)*, pp. 611–629. Dordrecht: Kluwer Academic Publishers.

CAMPBELL, M. I., CAGAN, J. & KOTOVSKY, K. (1998). A-Design: theory and implementation of an adaptive, agent-based method of conceptual design. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '98 (AID '98)*, pp. 579–598. Dordrecht: Kluwer Academic Publishers.

CANDY, L. (1997). Computers and creativity support: knowledge, visualisation and collaboration. In K. HORI, Ed., *Knowledge-Based Systems*, Special Issue on Information Technology Support for Creativity, **10**(1):3–13.

CANDY, L. (1998). Representations of strategic knowledge in design. In K. HORI, Ed., *Knowledge-Based Systems*, Special Issue on Strategic Knowledge and Concept Formation, **11**(7-8):379–390.

CANDY, L. & EDMONDS, E. (1994). Artefacts and the designer's process: implications for computer support to design. *Revue Sciences et Techniques de la Conception*, **3**(1):11–31.

CANDY, L. & EDMONDS, E. (1996). Creative design of the Lotus bicycle: implications for knowledge support systems research. *Design Studies*, **17**(1):71–90.

CANDY, L., EDMONDS, E. & PATRICK, D. J. (1995). Interactive support to conceptual design, AI system support for conceptual design. In: J. SHARPE, Ed., *Proc. Lancaster Int. Workshop on Engineering Design*, pp. 260–278. Berlin: Springer-Verlag.

CHANDRASEKARAN, B. (1990). Design problem solving: a task analysis. *AI Magazine*, **11**(4):59–71.

CHARLTON, C. & WALLACE, K. (2000). Reminding and context in design. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '00 (AID '00)*, pp. 569–588. Dordrecht: Kluwer Academic Publishers.

CHUNG, P. W. H. & GOODWIN, R. (1994). Representing design history. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '94 (AID '94)*, pp. 735–752. Dordrecht: Kluwer Academic Publishers.

CONSOLE, L. & TORASSO, P. (1990). Hypothetical reasoning in causal models. *Int. J. Intelligent Systems*, **5**(1):83–124.

CORNELISSEN, F., JONKER, C. M. & TREUR, J. (2002). Compositional verification of knowledge-based task models and problem solving methods. *Knowledge and Information Systems Journal*, to appear.

COYNE, R. D., ROSENMAN, M. A., RADFORD, A. D., BALACHANDRAN, M. & GERO, J. S. (1990). *Knowledge-Based Design Systems.* Reading: Addison-Wesley.

DAVIS, A. M. (1993). *Software Requirements: Objects, Functions and States*. New Jersey: Prentice Hall.

DUNSKUS, B. V., GRECU, D. L., BROWN, D. C. & BERKER, I. (1995). Using single function agents to investigate conflict. In I. F. C. SMITH, Ed., *AIEDAM*, Special Issue on Conflict Management in Design, **9**(4):299–312.

ENGELFRIET, J., JONKER, C. M. & TREUR, J. (2002). Compositional verification of multi-agent systems in temporal multi-epistemic logic. *J. Logic, Language and Information*, **11**:195–225.

ENGELFRIET, J. & TREUR, J. (1994). Temporal theories of reasoning. In C. MACNISH, D. PEARCE & L. M. PEREIRA, Eds., *Proc. Fourth Europ. Workshop on Logics in Artificial Intelligence (JELIA '94)*, Lecture Notes in Computer Science No. 838, pp. 279–299. Berlin: Springer-Verlag.

FENSEL, D. (1993). *The Knowledge Acquisition and Representation Language KARL*. Ph.D. Thesis. Karlsruhe: University of Karlsruhe.

FENSEL, D., ANGELE, J. & LANDES, D. (1991). KARL: a knowledge acquisition and representation language. *Proc. Eleventh Int. Conf. on Artificial Intelligence, Expert Systems and Natural Language (Avignon-91)*, **1**:513–525. Nanterre: EC2.

FISCHER, G. & NAKAKOJI, K. (1997). Computational environments supporting creativity in the context of lifelong learning and design. In K. HORI, Ed., *Knowledge-Based Systems*, Special Issue on Information Technology Support for Creativity, **10**(1):21–28.

FISCHER, G., NAKAKOJI, K., OSTWALD, J., STAHL, G. & SUMNER, T. (1993). Embedding critics in design environments. *The Knowledge Engineering Review*, **4**(8):285–307.

FRENCH, M. J. (1985). *Conceptual Design for Engineers*. London: The Design Council.

GANESHAN, R., GARRET, J. & FINGER, S. (1994). A framework for representing design intent. *Design Studies*, **15**(1):59–84.

GEELEN, P. A. & KOWALCZYK, W. (1992). A knowledge-based system for the routing of international blank payment orders. *Proc. Twelfth Int. Conf. on Artificial Intelligence, Expert Systems and Natural Language (Avignon-92)*, **2**:669–677. Nanterre: EC2.

GEELEN, P.A., RUTTKAY, ZS. & TREUR, J. (1992). On the (non-)brittleness of a DESIRE-model. In M. LINSTER, Ed., Sisyphus '92: Models of Problem Solving. Sisyphus Yearbook 1992, *GMD Working Papers 630*.

GERO, J. S. (1990). Design prototypes: a knowledge representation schema for design. *AI Magazine*, **11**(4):26–36.

GERO, J. S. (1998). Concept formation in design. In K. HORI, Ed., *Knowledge-Based Systems*, Special Issue on Strategic Knowledge and Concept Formation, **11**(7-8):429–435.

GOEL, A. & CHANDRASEKARAN, B. (1989). Functional representation of design and redesign problem solving. In N. S. SRIDHARAN, Ed., *Proc. Eleventh Int. Joint Conf. on Artificial Intelligence (IJCAI '89)*, pp. 1388–1394. Los Altos: Morgan Kaufmann.

GOEL, V. & PIROLLI, P. (1989). Motivating the notion of generic design within information-processing theory: the design problem space. *AI Magazine*, Spring, pp. 18–36.

GRECU, D. L. & BROWN, D. C. (1996). Learning by single function agents during spring design. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '96 (AID '96)*, pp. 409–428. Dordrecht: Kluwer Academic Publishers.

GRUBER, T., BAUDIN, C., BOOSE, J. & WEBER, J. (1991). Design rationale capture as knowledge acquisition: trade-offs in the design of interactive tools. *Proc. Eighth Int. Workshop on Machine Learning*, pp. 3–12. Chicago: Morgan Kaufmann.

GRUBER, T. R., OLSEN, G. R. & RUNKEL, J. (1993). The configuration design ontologies and the VT elevator domain theory. In A. TH. SCHREIBER & W. P. BIRMINGHAM, Eds., *Int. J. Human-Computer Studies*, Special Issue on Sisyphus-VT, **44**:569–598.

GRUBER, T. & RUSSEL, D. M. (1990). *Design knowledge and design rationale: a framework for representation, capture and use*. Technical Report KSL 90-45. Stanford, CA: Stanford University, Knowledge Systems Laboratory.

HAMMOND, P., DAVENPORT, J. C. & FITZPATRICK, F. J. (1993). Logic-based integrity constraints and the design of dental prostheses. *Artificial Intelligence in Medicine*, **9**:431–446.

HAROUD, D., BOULANGER, S., GELLE, E. & SMITH, I. F. C. (1994). Management of conflict for preliminary engineering design tasks. In I. F. C. SMITH, Ed., *AIEDAM*, Special Issue on Conflict Management in Design, **9**(4):313–323.

HEDTSTÜCK, U. & SCHMITT, P.H. (1989). A calculus for order-sorted predicate logic with sort literals. In K. H. BLÄSIUS, U. HEDTSTÜCK & C.-R. ROLLINGER, Eds., *Proc. Workshop on Sorts and Types in Artificial Intelligence*, Lecture Notes in Artificial Intelligence No. 418, pp. 61–72. Berlin: Springer-Verlag.

HERLEA, D. E., JONKER, C. M., TREUR, J. & WIJNGAARDS, N. J. E. (1999*a*). Integration of behavioural requirements specification within knowledge engineering. In: D. FENSEL & R. STUDER, Eds., *Knowledge Acquisition, Modelling and Management, Proc. Eleventh Europ. Workshop on Knowledge Acquisition, Modelling and Management (EKAW '99)*, Lecture Notes in Artificial Intelligence No. 1621, pp. 173–190. Berlin: Springer-Verlag.

HERLEA, D. E., JONKER, C. M., TREUR, J. & WIJNGAARDS, N. J. E. (1999*b*). Specification of behavioural requirements within compositional multi-agent system design. In: F. J. GARIJO & M. BOMAN, Eds., *Multi-Agent System Engineering, Proc. Ninth Europ. Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW '99)*, Lecture Notes in Artificial Intelligence No. 1647, pp. 8–27. Berlin: Springer-Verlag.

HORI, K. (1997). Where is, what is and how can we use strategic knowledge? In L. CANDY & K. HORI, Eds., *Proc. First Int. Workshop on Strategic Knowledge and Concept Formation*, pp. 35–42. Loughborough: Loughborough University of Technology.

JONKER, C. M. & TREUR, J. (1998). Compositional verification of multi-agent systems: a formal analysis of pro-activeness and reactiveness. In: W. P. DE ROEVER, H. LANGMAACK & A. PNUELI, Eds., *Proc. Int. Workshop on Compositionality (COMPOS '97)*, Lecture Notes in Computer Science No. 1536, pp. 350–380. Berlin: Springer-Verlag. Extended version (2002) in: *Int. Journal of Cooperative Information Systems*, **11**:51–92.

KLEENE, S. C. (1952). *Introduction to Metamathematics*. Amsterdam: North-Holland.

KLEIN, M. (1992). DRCS: an integrated system for capture of designs and their rationale. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '92 (AID '92)*, pp. 393–412. Dordrecht: Kluwer Academic Publishers.

KLEIN, M. (1995). Conflict management as part of an integrated exception handling approach. In I. F. C. SMITH, Ed., *AIEDAM*, Special Issue on Conflict Management in Design, **9**(4):259–267.

KLEIN, R. (2000). Knowledge modelling in design – the MOKA framework. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '00 (AID '00)*, pp. 77–102. Dordrecht: Kluwer Academic Publishers.

KOLLER, R. (1985). *Konstruktionslehre für den Maschinenbau*. Berlin: Springer-Verlag.

LANDES, D. (1994). DesignKARL – a language for the design of knowledge-based systems. Extended version of earlier paper which appeared in: *Proc. Sixth Intern. Conf. on Software and Knowledge Engineering (SEKE '94)*, pp. 78–85.

LANGEN, P. H. G. VAN, BRAZIER, F. M. T., DIEPENMAAT, H. B. & PULLES, M. P. J. (1995). *Inventarisatie van de milieubelasting van industriële processen vanuit kennistechnologisch perspectief.* Technical Report TNO-MW-R95/139. Delft: TNO Environmental Sciences.

LANGEN, P. H. G. VAN & TREUR, J. (1989). *Representing world situations and information states by many-sorted partial models*. Technical Report PE8904. Amsterdam: University of Amsterdam, Programming Research Group, Expert Systems Section.

LANGHOLM, T. (1988). *Partiality, Truth and Persistence.* CSLI Lecture Notes No. 15. Stanford: Stanford University.

LAWSON, B. (1997). *How Designers Think*. Completely revised third edition. Oxford: Architectural Press.

LEEMANS, P., TREUR, J. & WILLEMS, M. (2002). A semantical perspective on verification of knowledge. *Data and Knowledge Engineering*, **40**:33–70.

LÖCKENHOFF, C. & MESSER, T. (1994). Configuration. In J. A. P. J. BREUKER & W. VAN DE VELDE, EDS., *CommonKADS Library for Expertise Modelling: Reusable Problem Solving Components*, pp. 197–212. Amsterdam: IOS Press.

LOGAN, B. S. & SMITHERS, T. (1992). Creativity and design as exploration. In J. S. GERO & M. L. MAHER, Eds., *Modelling Creativity and Knowledge-Based Creative Design*, pp. 149–188. Hillsdale: Lawrence Erlbaum.

MAES, P. (1987). *Computational Reflection*. Ph.D. Thesis, Technical Report 87–2. Brussels: Vrije Universiteit Brussel, AI Lab.

MAES, P. (1988). Issues in computational reflection. In P. MAES & D. NARDI, Eds., *Meta-level Architectures and Reflection*, pp. 21–35. Amsterdam: Elsevier (North-Holland).

MAHER, M. L. (1990). Process models for design synthesis. *AI Magazine*, **4**(1):49–58.

MARCUS, S. & MCDERMOTT, J. (1989). SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, **39**:1–37.

MARCUS, S., STOUT, J. & MCDERMOTT, J. (1988). VT: An expert elevator designer that uses knowledge-directed backtracking. *AI Magazine*, **9**:95–112.

MCALINDEN, L. P., FLORIDA-JAMES, B. O., CHAO, K.-M., NORMAN, P. W., HILLS, W. & SMITH, P. (1998). Information and knowledge sharing for distributed design agents. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '98 (AID '98)*, pp. 537–556. Dordrecht: Kluwer Academic Publishers.

MCKERLIE, D. & MCLEAN, A. (1994). Reasoning with design rationale: practical experience with design space analysis. *Design Studies*, **15**(2):214–226.

MOSTOW, J. (1985). Toward better models of the design process. *AI Magazine*, **6**(1):44–57.

MOSTOW, J. (1989). Design by derivational analogy: issues in the automated replay of design plans. *Artificial Intelligence*, **40**:119–184.

MOTTA, E., STUTT, A., ZDRAHAL, Z., O'HARA, K. & SHADBOLT, N. (1996). Solving VT in VITAL: a study in model construction and knowledge reuse. In A. TH. SCHREIBER & W. P. BIRMINGHAM, Eds., *Int. J. Human-Computer Studies,* Special Issue on Sisyphus-VT, **44**:333–371.

NAKAKOJI, K., SUMNER, T. & HARSTAD, B. (1994). Perspective-based critiquing: helping designers cope with conflicts among design intentions. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '94 (AID '94)*, pp. 449–466. Dordrecht: Kluwer Academic Publishers.

NEWELL, A. (1982). The knowledge level. *Artificial Intelligence*, **18**:87–127.

OH, V. & SHARPE, J. (1995). Conflict management in an interdisciplinary design environment. In I. F. C. SMITH, Ed., *AIEDAM*, Special Issue on Conflict Management in Design, **9**(4):247–258.

OHSUGA, S. (1997). Strategic knowledge makes knowledge based systems truly intelligent. In L. CANDY & K. HORI, Eds., *Proc. First Int. Workshop on Strategic Knowledge and Concept Formation*, pp. 1–24. Loughborough: Loughborough University of Technology.

PAHL, G. & BEITZ, W. (1984). *Engineering design*. New York: Springer-Verlag.

PETRIE, C. J. (1992). Constrained decision revision. *Proc. Tenth National Conference on AI*, pp. 393–400. San Jose: AAAI Press.

PETRIE, C. J., CUTKOSKY, M. R. & PARK, H. (1994). Design space navigation as a collaborative aid. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '94 (AID '94)*, pp. 611–623. Dordrecht: Kluwer Academic Publishers.

PETRIE, C. J., WEBSTER, T. A. & CUTKOSKY, M. R. (1995). Using Pareto optimality to coordinate distributed agents. In I. F. C. SMITH, Ed., *AIEDAM*, Special Issue on Conflict Management in Design, **9**(4):269–281.

POECK, K., FENSEL, D., LANDES, D. & ANGELE, J. (1996). Combining KARL and CRLM for designing vertical transportation systems. In A. TH. SCHREIBER & W. P. BIRMINGHAM, Eds., *Int. J. Human-Computer Studies*, Special Issue on Sisyphus-VT, **44**:435–467.

PROTZEN, J.-P., HARRIS, D. & CAVALLIN, H. (2000). Limited computation, unlimited design. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '00 (AID '00)*, pp. 43–52. Dordrecht: Kluwer Academic Publishers.

PUGH, S. (1990). *Total Design: Integrated Methods for Successful Product Engineering*. Reading: Addison-Wesley.

REFFAT, R. & GERO, J. (2000). Computational situated learning in design – Application to architectural shape semantics. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '00 (AID '00)*, pp. 589–610. Dordrecht: Kluwer Academic Publishers.

REITER, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, **32**:57–96.

RITTEL, H. & WEBBER, M. (1969). Dilemmas in the general theory of planning. *DMG-DRS Journal*, **8**:219–233.

ROTHENFLUH, T. E., GENNARI, J. H., ERIKSSON, H., PUERTA, A. R., TU, S. W. & MUSEN, M. A. (1996). Reusable ontologies, knowledge-acquisition tools, and performance systems: PROTÉGÉ-II solutions to Sisyphus-2. In A. TH. SCHREIBER & W. P. BIRMINGHAM, Eds., *Int. J. Human-Computer Studies*, Special Issue on Sisyphus-VT, **44**:303–332.

RUNKEL, J. T., BALKANY, A. & BIRMINGHAM, W. P. (1994). Generating non-brittle configuration-design tools. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '94 (AID '94)*, pp. 183–200. Dordrecht: Kluwer Academic Publishers.

RUNKEL, J. T., BIRMINGHAM, W. P. & BALKANY, A. (1996). Solving VT by reuse. In A. TH. SCHREIBER & W. P. BIRMINGHAM, Eds., *Int. J. Human-Computer Studies*, Special Issue on Sisyphus-VT, **44**:403–433.

SCHÖN, D. A. (1983). *The Reflective Practitioner: How Professionals Think in Action*. London: Temple Smith.

SCHREIBER, A. TH. & TERPSTRA, P. (1996). Sisyphus-VT: a CommonKADS solution. In A. TH. SCHREIBER & W. P. BIRMINGHAM, Eds., *Int. J. Human-Computer Studies*, Special Issue on Sisyphus-VT, **44**:373–402.

SCHREIBER, A. TH., WIELINGA, B. J., HOOG, R. DE, AKKERMANS, H. & VAN DE VELDE, W. (1994). CommonKADS: a comprehensive methodology for KBS development. *IEEE Expert*, **9**(6):28–37.

SIM, S. K. & DUFFY, A. H. B. (2000). Evaluating a model of learning in design using protocol analysis. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '00 (AID '00)*, pp. 455–477. Dordrecht: Kluwer Academic Publishers.

SIMON, H. A. (1973). The structure of ill-structured problems. *Artificial Intelligence*, **4**:181–201.

SMITHERS, T. (1996). On knowledge level theories of design process. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '96 (AID '96)*, pp. 561–579. Dordrecht: Kluwer Academic Publishers.

SMITHERS, T. (1998). Towards a knowledge level theory of design process. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '98 (AID '98)*, pp. 3–21. Dordrecht: Kluwer Academic Publishers.

SMITHERS, T., CORNE, D. & ROSS, P. (1994). On computing exploration and solving design problems. In J. S. GERO & E. TYUGU, Eds., *Formal Design Methods for CAD*, pp. 293–313. Amsterdam: Elsevier.

SOMMERVILLE, I. & SAWYER, P. (1997). *Requirements Engineering: A Good Practice Guide*. Chichester: John Wiley & Sons.

STOLZE, M. (1994). Visual critiquing in domain oriented design environments: showing the right thing at the right place. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '94 (AID '94)*, pp. 467–482. Dordrecht: Kluwer Academic Publishers.

STRELNIKOV, Y. N. & DMITREVICH, G. D. (1991). Formal description and comparison of interactive design strategies. *Artificial Intelligence in Engineering*, **6**(4):186–195.

SUH, N. P. (1990). *The Principles of Design*. Oxford Series of Advanced Manufacturing. Oxford: Oxford University Press.

SUMI, Y., HORI, K. & OHSUGA, S. (1998). Supporting the acquisition and modeling of requirements in software design. In K. HORI, Ed., *Knowledge-Based Systems*, Special Issue on Strategic Knowledge and Concept Formation, **11**(7-8):449–456.

SUN, K. & FALTINGS, B. (1994). Supporting creative mechanical design. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '94 (AID '94)*, pp. 39–56. Dordrecht: Kluwer Academic Publishers.

TAKEDA, H. & NISHIDA, T. (1994). Integration of aspects in design processes. In J. S. GERO & F. SUDWEEKS, Eds., *Proc. Artificial Intelligence in Design '94 (AID '94)*, pp. 309–326. Dordrecht: Kluwer Academic Publishers.

TAKEDA, H., VEERKAMP, P. J., TOMIYAMA, T. & YOSHIKAWA, H. (1990). Modelling design processes. *AI Magazine*, **11**(4):37–48.

TOMIYAMA, T. & YOSHIKAWA, H. (1987). Extended general design theory. In H. YOSHIKAWA & E. A. WARMAN, Eds., *Proc. IFIP WG5.2 Working Conf. on Design Theory for CAD*, pp. 95–125. Amsterdam: Elsevier (North-Holland).

TONG, T. (1987). Toward an engineering science of knowledge-based design. *Artificial Intelligence in Engineering*, **2**(3):133–166.

TREUR, J. (1989). A logical analysis of design tasks for expert systems. *Int. J. Expert Systems*, **2**(2):233–253.

TREUR, J. (1991). A logical framework for design processes. In P. J. W. TEN HAGEN & P. J. VEERKAMP, Eds., *Intelligent CAD Systems III*. Proc. Third Eurographics Workshop on Intelligent CAD Systems, pp. 3–19. Berlin: Springer-Verlag.

TREUR, J. (1994). Temporal semantics of meta-level architectures for dynamic control of reasoning. In L. FRIBOURG & F. TURINI, Eds., *Proc. Fourth Int. Workshop on Meta-Programming in Logic (META '94)*, Lecture Notes in Computer Science No. 883, pp. 353–376. Berlin: Springer-Verlag. Extended version (2002) in: *Int. J. Information Systems*, **17**:545–568.

TREUR, J. & WILLEMS, M. (1994). A logical foundation for verification. In A. G. COHN, Ed., *Proc. 11th Europ. Conf. on Artificial Intelligence (ECAI '94)*, pp. 745–749. Chichester: John Wiley & Sons.

URSU, M. F. & HAMMOND, P. (2000). Expressing regulatory design knowledge for critiquing intelligent design assistants. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '00 (AID '00)*, pp. 103–126. Dordrecht: Kluwer Academic Publishers.

VANWELKENHUYSEN, J. & MIZOGUCHI, R. (1995). Workplace-adapted behaviours: lessons learned for knowledge reuse. In N.J.I. MARS, Ed., *Towards Very Large Knowledge Bases*, Proc. 2nd Int. Conf. on Building and Sharing Very Large-Scale Knowledge Bases, pp. 270–280. Amsterdam: IOS Press.

VAREJÃO, F. M., DE MENEZES, C. S., GARCIA, A. C. B., DE SOUZA, C. S. & FROMHERZ, M. P. J. (2000). Towards an ontological framework for knowledge-based design systems. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '00 (AID '00)*, pp. 55–75. Dordrecht: Kluwer Academic Publishers.

WALLAS, G. (1926). *The Art of Thought*. New York: Harcort Brace.

WEYHRAUCH, R. W. (1980). Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, **13**:133–170.

WIELINGA, B. J., AKKERMANS, J. M. & SCHREIBER, A. TH. (1995). A formal analysis of parametric design. In B. R. GAINES & M. A. MUSEN, Eds., *Proc. Ninth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW '95)*, **2**:37/1–37/15. Calgary: University of Calgary, Department of Computer Science, SRDG Publications.

WIELINGA, B. J. & SCHREIBER, A. TH. (1997). Configuration-design problem solving. *IEEE Expert*, **12**(2):49–56.

WOOLDRIDGE, M. J. & JENNINGS, N. R. (1995). Intelligent agents: theory and practice. *Knowledge Engineering Review*, **10**(2):115–152.

YOST, G. R. (1996). Implementing the Sisyphus-93 task using Soar/TAQL. In A. TH. SCHREIBER & W. P. BIRMINGHAM, Eds., *Int. J. Human-Computer Studies*, Special Issue on Sisyphus-VT, **44**:281–301.

YOST, G. R. & ROTHENFLUH, T. E. (1996). Configuring elevator systems. In A. TH. SCHREIBER & W. P. BIRMINGHAM, Eds., *Int. J. Human-Computer Studies*, Special Issue on Sisyphus-VT, **44**:521–568.

ZHAO, F. & MAHER, M. L. (1992). Using network-based prototypes to support creative design by analogy and mutation. In J. S. GERO, Ed., *Proc. Artificial Intelligence in Design '92 (AID '92)*, pp. 773–793. Dordrecht: Kluwer Academic Publishers.

Samenvatting

De Anatomie van het Ontwerpen: Grondslagen, Modellen en Toepassingen

De wereld van vandaag is ondenkbaar zonder de objecten die mensen hebben gemaakt. Aan deze objecten kennen we allerlei functies toe: huizen zijn er typisch om in te wonen, vliegtuigen om mee te vliegen en computersoftware om bijvoorbeeld mee te handelen op de beurs. Doelbewuste creatie van een object vereist dat er wordt nagedacht over de *eisen* die aan het object worden gesteld en het *gedrag*, de *structuur* en de *vorm* die het object moet hebben zodat het aan deze eisen voldoet. Deze uitgangssituatie is typisch voor een *ontwerpproces*.

Ontwerpprocessen worden in de praktijk vaak genoemd naar hun toepassingsdomeinen. Zo heeft *architectuur* betrekking op het ontwerpen van gebouwen, *werktuigbouwkunde* op het ontwerpen en bouwen van werktuigen (hulpmiddelen, gereedschappen en machines) en *industrieel ontwerpen* op het ontwerpen van industriële producten. Ook culturele activiteiten zoals beleidsvorming, politiek en andere vormen van beïnvloeding van het menselijk gedrag worden steeds vaker als ontwerpprocessen gezien. Iemand die zich professioneel bezighoudt met ontwerpen wordt een *ontwerper* genoemd (of naar het toepassingsdomein: architect etc.).

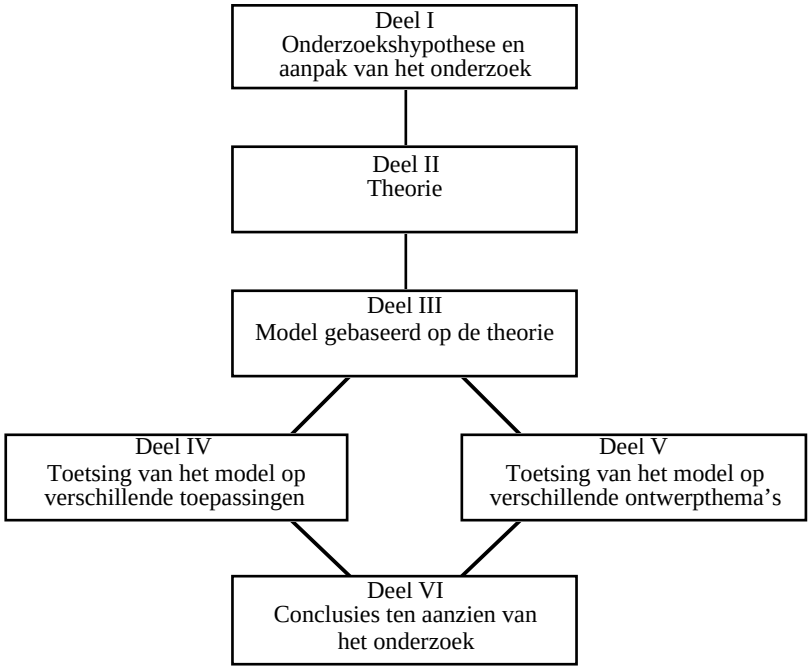
In de praktijk wordt een ontwerper geconfronteerd met problematische situaties die raadselachtig, verontrustend, conflicterend en onzeker kunnen zijn. Dit vloeit voort uit het feit dat de behoeften en wensen die de ontwerper moet zien te vervullen, en het ontwerpprobleem dat hij/zij daaruit aanvankelijk afleidt, meestal *slecht gestructureerd* zijn. Dit betekent dat zo'n ontwerpprobleem niet geheel kan worden herleid een bekende probleemcategorie: de functie van het te ontwerpen object, de omgeving waarin het object deze functie moet bieden of de combinatie hiervan is uniek. Hierdoor lukt het niet om dit ontwerpprobleem in zijn geheel op te lossen met de voor bekende probleemcategorieën beschikbare methoden en technieken.

Dit proefschrift beoogt een bijdrage te leveren aan het wetenschapsgebied dat (meestal ook door Nederlandstalige onderzoekers) wordt aangeduid als *AI in Design* (voluit: *Artificial Intelligence in Design*). Dit vakgebied bevindt zich op het snijvlak van de kunstmatige intelligentie, informatica, cognitieve psychologie, logica, wiskunde, architectuur en werktuigbouwkunde. De bijdrage bestaat uit het bieden van een basis voor een beter begrip van praktische ontwerpprocessen en voor de ontwikkeling van ontwerpondersteunende systemen. Een ontwerper die ontwerpprocessen beter doorgrondt en zich ondersteund weet door bruikbare hulpmiddelen is hopelijk beter in staat zijn of haar ontwerpprestaties te verbeteren.

Het proefschrift bestaat inhoudelijk uit zes delen, die samen zestien hoofdstukken vormen:

- Deel I: Een voorstel voor onderzoek op het gebied van *AI in Design*
- Deel II: Grondslagen van het ontwerpen
- Deel III: Een generiek model van het ontwerpen
- Deel IV: Ontwerptoepassingen
- Deel V: Ontwerpthema's
- Deel VI: Conclusies en verder onderzoek

Figuur S.1 geeft schematisch aan hoe deze delen in het onderzoek passen en hoe ze samenhangen.



FIGUUR S.1. Schematisch overzicht van het proefschrift.

Een voorstel voor onderzoek op het gebied van *AI in Design*

In de praktijk houdt het ontwerpen in dat moet worden bepaald wat nu precies het probleem is dat moet worden opgelost om in de behoeften en wensen van een klant of opdrachtgever te voorzien. Dit vindt soms plaats voordat, maar meestal terwijl aan een beschrijving van een ontwerpobject wordt gewerkt. Door deze beschrijving te maken komt de ontwerper op ideeën over wat het ontwerpprobleem nu eigenlijk is. Deze met vele onderzoekers gedeelde optiek vormt de basis voor de theorie, de modellen en de toepassingen in dit proefschrift.

Gegeven de slechte structuur van ontwerpproblemen rijst de vraag of een ontwerpproces dan ook per definitie slecht gestructureerd is. In dit proefschrift wordt de stelling opgeworpen dat dit niet het geval is. De centrale hypothese van dit proefschrift luidt als volgt: *Ook al is een (initieel) ontwerpprobleem typisch slecht gestructureerd, de hoofdstructuur (oftewel: de* anatomie*) van een ontwerpproces is welgedefinieerd en zelfs generiek, dat wil zeggen onafhankelijk van het toepassingsdomein en de toegepaste ontwerpmethode en -technieken.*

Uit onderzoek is nog relatief weinig bekend over de anatomie van ontwerpprocessen. Veel aandacht gaat uit naar representaties, methoden, techieken en systemen, maar de theorie is vaak onderbelicht. Bovendien wordt geen van de bestaande theorieën als geheel bevredigend gezien.

Om te proberen in deze leemte te voorzien is het onderzoek als volgt aangepakt. Door middel van logica is een analyse van ontwerpprocessen op het kennisniveau gemaakt. Dit is het door Newell gedefinieerde niveau waarop kennis wordt verwerkt, doelen en acties worden onderscheiden, complexe kennis uit eenvoudige kennis kan worden opgebouwd en wordt uitgegaan van rationeel handelen. Via deze analyse is gepoogd zicht te krijgen op de processen en de kennis die een rol spelen bij het bepalen van het op te lossen ontwerpprobleem, het bedenken van een ontwerpoplossing, het omgaan met deadlines, budgettaire beperkingen etc. en de interactie tussen al deze activiteiten (met wederzijdse beïnvloeding tot gevolg).

Omdat ontwerpen een denkproces is, kan niet zomaar worden bewezen dat zo'n analyse correct of volledig is; hiervan kan alleen een inschatting worden gemaakt door theorie en praktijk met elkaar te confronteren. Deze confrontatie bestaat uit het herhaaldelijk bestuderen van in de literatuur beschreven ontwerpprocessen, observeren van ontwerpprocessen in de praktijk, vormen van een theorie over de anatomie van ontwerpprocessen en toetsen van de gevormde theorie aan de praktijk.

Grondslagen van het ontwerpen

Eén van de hoofdresultaten van het onderzoek is een logische (d.w.z. op logica gebaseerde) theorie, die de anatomie van ontwerpprocessen beschrijft. De logische theorie betreft zowel de statische aspecten als de dynamische aspecten van ontwerpprocessen. Voor beide soorten aspecten geldt dat hierbij concepten en relaties tussen concepten zijn gedefinieerd.

De statische aspecten van ontwerpprocessen betreffen de verschillende typen beschrijvingen die een rol spelen in ontwerpprocessen, zoals een *ontwerpeisenpakket* of een *ontwerpobjectbeschrijving*. Voorbeelden van relaties tussen zulke beschrijvingen zijn *consistentie* van twee ontwerpobjectbeschrijvingen en de *vervulling* van een ontwerpeisenpakket door een specifieke ontwerpobjectbeschrijving.

De dynamische aspecten van ontwerpprocessen betreffen de verschillende typen toestanden en reeksen van toestandsovergangen die een rol spelen in ontwerpprocessen, zoals het *wijzigen van de huidige ontwerpobjectbeschrijving* of het *verfijnen van het huidige ontwerpeisenpakket*. Een voorbeeld van een relatie tussen ontwerptoestanden en reeksen van toestandsovergang is een *ontwerpstrategie* die bijvoorbeeld voorschrijft dat eerst een definitief ontwerpeisenpakket moet worden bepaald voordat aan het genereren van een ontwerpobjectbeschrijving mag worden begonnen.

Om de dynamiek van een ontwerpproces gestructureerd te kunnen beschrijven worden *reflectieniveaus* onderscheiden, waarbij een hoger niveau redeneert over het redeneerproces dat op het niveau direct eronder plaatsvindt, met als doel hieraan sturing te geven:

- Het *object-niveau* is het basisniveau, dat informatie en kennis bevat over objecten en hun onderlinge relaties in het toepassingsdomein.
- Het *eerste meta-niveau* bevat informatie en kennis over ontwerpobjectbeschrijvingen, individuele ontwerpeisen en hun (onderlinge) relaties.
- Het *tweede meta-niveau* bevat informatie en kennis over ontwerpeisenpakketten, hun onderlinge relaties en de relaties met ontwerpobjectbeschrijvingen, naast informatie en kennis over het wijzigen van ontwerpeisenpakketten en ontwerpobjectbeschrijvingen.
- Het *derde meta-niveau* bevat informatie en kennis over de doelstellingen van het ontwerpproces (zoals een deadline), ontwerpstrategieën en evaluaties van het ontwerpproces.

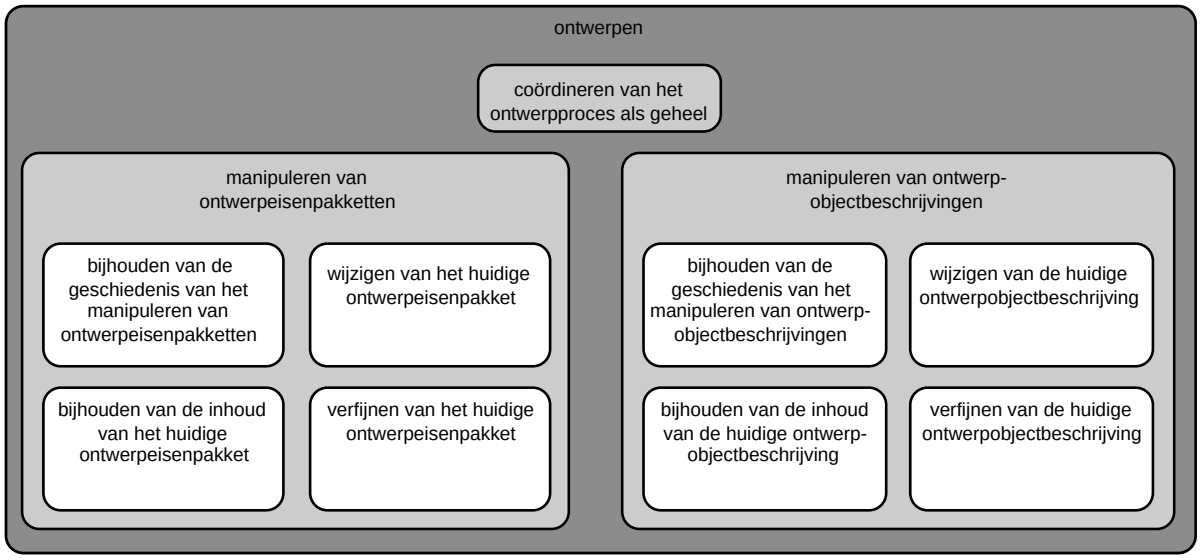
Een generiek model van het ontwerpen

Een ander hoofdresultaat van het onderzoek is een generiek ontwerpmodel genaamd GDM, dat is gemaakt op basis van de eerdergenoemde logische theorie. GDM biedt een specificatie die rechtstreeks kan worden omgezet naar computersoftware. GDM is (hierdoor) op sommige punten wel wat beperkter dan de logische theorie. Het is bijvoorbeeld in de logische theorie eenvoudig om te stellen dat als er geen ontwerpobjectbeschrijving bestaat die een specifieke ontwerpeis vervult, deze ontwerpeis dan dient te worden verwijderd uit het huidige ontwerpeisenpakket; deze stelregel is op een computer echter niet te hanteren, gezien de grootte van de ruimte van alle mogelijke ontwerpobjectbeschrijvingen.

De componentgebaseerde methode die is gebruikt om GDM te ontwikkelen heet DESIRE. Met DESIRE (ontwikkeld door de afdeling Kunstmatige Intelligentie van de Vrije Universiteit) kunnen modellen van complexe redeneertaken (zoals ontwerptaken) en agentsystemen

(zoals mobiele agenten op het Internet) worden ontwikkeld. DESIRE definieert een logische taal op het kennisniveau, die bij uitstek geschikt is om de statische aspecten en dynamische aspecten van ontwerpprocessen te modelleren.

GDM bestaat uit in totaal drie procesabstractieniveaus. Op elk niveau worden processen gedefinieerd in termen van invoer en uitvoer, evenals de uitwisseling van informatie en de besturing van deze processen. Figuur S.2 toont de procesabstractieniveaus en de processen die op deze niveaus in GDM worden onderkend.



FIGUUR S.2. Processen op de drie procesabstractieniveaus van GDM.

GDM's hoogste procesabstractieniveau bevat een enkele component voor het ontwerpproces als geheel. Het op een na hoogste procesabstractieniveau bestaat uit een drietal deelcomponenten voor:

- het manipuleren van ontwerpeisenpakketten,
- het manipuleren van ontwerpobjectbeschrijvingen,
- het coördineren van het ontwerpproces als geheel.

GDM's deelcomponent voor de manipulatie van ontwerpeisenpakketten bestaat uit een viertal deelcomponenten, waaronder het verfijnen van het huidige ontwerpeisenpakket (bijvoorbeeld het aanvullen van ontwerpeisen die door een opdrachtgever aan een gebouw worden gesteld met eisen op het gebied van brandveiligheid). Het bijhouden van de geschiedenis van het manipuleren van ontwerpeisenpakketten gebeurt onder meer om indien gewenst een eerdere wijzigingsbeslissing te kunnen herzien.

GDM's deelcomponent voor de manipulatie van ontwerpobjectbeschrijvingen bestaat uit een viertal deelcomponenten, waaronder het verfijnen van de huidige ontwerpobjectbeschrijving (bijvoorbeeld het afleiden van de bouwoppervlakte van een gebouw uit gegevens over het vloerplan). Verder wordt ook hier de geschiedenis bijgehouden, in dit geval van het manipuleren van ontwerpobjectbeschrijvingen, dat nodig is om bijvoorbeeld een eerder als oplossing verworpen ontwerpobjectbeschrijving opnieuw te kunnen bekijken.

Ontwerptoepassingen

Om de geldigheid van de logische theorie van ontwerpprocessen en de bruikbaarheid van GDM in de praktijk te beproeven, is in het onderzoek een specialisatie van GDM gemaakt en zijn op basis hiervan twee ontwerptoepassingen ontwikkeld. De specialisatie van GDM bestaat uit deelcomponenten voor het wijzigen van ontwerpeisenpakketten en voor het wijzigen van ontwerpobjectbeschrijvingen. Alhoewel niet generiek zijn deze specialisaties nuttig gebleken in vele praktische ontwerptoepassingen.

Eén toepassing van GDM betreft de liftconfiguratietaak VT. Dit is een speciaal type ontwerptaak, waarbij een lift het ontwerpobject vormt. De ontwerpeisen bestaan hier uit klantspecificaties en omgevingsfactoren, zoals te respecteren dimensies van het gebouw waarin de lift dient te worden geplaatst en randvoorwaarden.

Een andere toepassing van GDM betreft milieu-inventarisatie. Het te ontwerpen object is hier een procesmodel, waarmee de milieu-impact van een specifiek industrieel proces kan worden bepaald. De ontwerpeisen bestaan uit eisen aan de kwaliteit van het procesmodel, zoals die door de landelijke en gemeentelijke overheden zijn opgelegd.

Ontwerpthema's

Om de bruikbaarheid van GDM voor het modelleren van niet-domeingebonden aspecten van ontwerpprocessen te beproeven zijn drie ontwerpthema's bekeken die reguliere onderwerpen van onderzoek binnen het vakgebied *AI in Design* vormen: strategische ontwerpkennis, ontwerprationale en management van ontwerpconflicten.

Strategische kennis is onder andere van belang in interactieve ontwerpprocessen. In zo'n proces onderhouden een ontwerper en een ontwerpsysteem een dialoog over de (te voeren of gevoerde) strategie voor het ontwerpproces. Om dit type proces te ondersteunen moet een ontwerpondersteunend systeem kunnen redeneren met strategische kennis. Deze kennis dient overeen te komen met de soorten strategische interactie die kunnen plaatsvinden tussen de ontwerper en het systeem: over het ontwerpproces in zijn geheel, over het manipuleren van ontwerpeisenpakketten en over het manipuleren van ontwerpobjectbeschrijvingen. GDM blijkt goed te kunnen worden gebruikt om de bijbehorende strategische kennis te modelleren.

Ook de ontwerpprocesgeschiedenis en de *ontwerprationale* zijn relevant voor ontwerpondersteunende systemen: deze maken het immers mogelijk een gebruiker uitleg te geven over het verloop en de resultaten van het ontwerpproces. Het gaat hierbij met name om het kunnen motiveren van de in het proces genomen beslissingen: waarom zijn ze genomen, wat waren de alternatieven en wat waren de voors en tegens. Ook wordt door gebruik te maken van de ontwerpprocesgeschiedenis en de ontwerprationale het mogelijk om (delen van) eerdere ontwerpeisenpakketten en ontwerpobjectbeschrijvingen te hergebruiken.

GDM biedt een structuur waarmee verschillende soorten ontwerprationale kunnen worden onderscheiden en gebruikt. Zo kan bijvoorbeeld met GDM worden aangetoond op welke momenten en hoe de ontwerprationale een rol speelt bij het ontwerpen van een deel van een nieuw type vliegtuig op basis van het ontwerp van een bestaand type.

Ten slotte is het voor ontwerpondersteunende systemen van belang aandacht te schenken aan het managen van ontwerpconflicten. Conflicten zoals een contradictie tussen twee eisen zijn immers inherent aan het ontwerpen. De opvatting waarop het managen van ontwerpconflicten is gestoeld is dat het ontwerpen een gecoördineerd proces is van:

- het opsporen van ontwerpconflicten,
- het toekennen van prioriteiten aan het oplossen ervan,
- het oplossen van (zojuist of al eerder geconstateerde) ontwerpconflicten.

De complexe redeneerprocessen die hierbij een rol spelen zijn zeer dynamisch en niet-monotoon. Het laatste wil zeggen dat het oplossen van het ene conflict gemakkelijk leidt tot het introduceren van een ander conflict.

Met GDM kan het managen van verschillende soorten ontwerpconflicten goed worden gemodelleerd. Hiervoor is het wel nodig specialisaties van GDM te maken.

Conclusies en verder onderzoek

Het onderzoek dat in dit proefschrift is beschreven heeft enkele bijdragen geleverd aan *AI in Design*, maar kent ook wat beperkingen. Een en ander is uit te splitsen naar ontwerptheorieën en -modellen, ontwerpmethoden en ontwerpondersteunende systemen.

Op het gebied van ontwerptheorieën en -modellen heeft het onderzoek geleid tot een nieuwe theorie, die op het kennisniveau beschrijft hoe ontwerpprocessen in elkaar steken. Ook heeft het onderzoek geleid tot GDM, een generiek model van ontwerpprocessen dat op veel gebieden een vooruitgang betekent ten opzichte van de in de literatuur bekende modellen. Zo onderkent geen enkel ander model ontwerpprocesdoelstellingen (zoals een beperking aan de duur van een ontwerpproces) en zijn de meeste modellen niet erg gedetailleerd met betrekking tot het manipuleren van ontwerpeisenpakketten en het manipuleren van ontwerpobjectbeschrijvingen. De logische theorie en GDM vormen duidelijke aanwijzingen dat de hypothese klopt, al vormen ze geen bewijs.

Aanbevolen wordt verder onderzoek te doen naar de anatomie van ontwerpprocessen. Zo is bijvoorbeeld het coördineren van een ontwerpproces als geheel relatief onderbelicht gebleven. Ook zaken die in het proefschrift niet of nauwelijks aan bod zijn gekomen, zoals creativiteit en gedistribueerd ontwerpen, zijn de moeite waard te worden onderworpen aan verder anatomisch onderzoek. Tevens is het zinvol onderzoek te verrichten naar specialisaties van GDM die van pas kunnen komen bij de analyse van specifieke ontwerpmethoden en toepassingsdomeinen, en naar applicatieontwikkelomgevingen waarmee op basis van (een bibliotheek van) specialisaties van GDM een bruikbaar ontwerpondersteunend systeem kan worden gemaakt.

SIKS Dissertatiereeks

1998-1 Johan van den Akker (CWI). *DEGAS - An Active, Temporal Database of Autonomous Objects.*

1998-2 Floris Wiesman (UM). *Information Retrieval by Graphically Browsing Meta-Information.*

1998-3 Ans Steuten (TUD). *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective.*

1998-4 Dennis Breuker (UM). *Memory versus Search in Games.*

1998-5 Eduard Oskamp (UL). *Computerondersteuning bij Straftoemeting.*

1999-1 Mark Sloof (VU). *Physiology of Quality Change Modelling: Automated modelling of Quality Change of Agricultural Products.*

1999-2 Rob Potharst (EUR). *Classification Using Decision Trees and Neural Nets.*

1999-3 Don Beal (UM). *The Nature of Minimax Search.*

1999-4 Jacques Penders (UM). *The Practical Art of Moving Physical Objects.*

1999-5 Aldo de Moor (KUB). *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems.*

1999-6 Niek Wijngaards (VU). *Re-design of Compositional Systems.*

1999-7 David Spelt (UT). *Verification Support for Object Database Design.*

1999-8 Jacques Lenting (UM). *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.*

2000-1 Frank Niessink (VU). *Perspectives on Improving Software Maintenance.*

2000-2 Koen Holtman (TUE). *Prototyping of CMS Storage Management.*

2000-3 Carolien Metselaar (UvA). *Sociaal-Organisatorische Gevolgen van Kennistechnologie: een Procesbenadering en Actorperspectief.*

2000-4 Geert de Haan (VU). *ETAG, a Formal Model of Competence Knowledge for User Interface Design.*

2000-5 Ruud van der Pol (UM). *Knowledge-based Query Formulation in Information Retrieval.*

2000-6 Rogier van Eijk (UU). *Programming Languages for Agent Communication.*

2000-7 Niels Peek (UU). *Decision-theoretic Planning of Clinical Patient Management.*

2000-8 Veerle Coupé (EUR). *Sensitivity Analyis of Decision-Theoretic Networks.*

2000-9 Florian Waas (CWI). *Principles of Probabilistic Query Optimization.*

2000-10 Niels Nes (CWI). *Image Database Management System Design Considerations: Algorithms and Architecture.*

2000-11 Jonas Karlsson (CWI). *Scalable Distributed Data Structures for Database Management.*

2001-1 Silja Renooij (UU). *Qualitative Approaches to Quantifying Probabilistic Networks.*

2001-2 Koen Hindriks (UU). *Agent Programming Languages: Programming with Mental Models.*

2001-3 Maarten van Someren (UvA). *Learning as Problem Solving.*

2001-4 Evgueni Smirnov (UM). *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets.*

2001-5 Jacco van Ossenbruggen (VU). *Processing Structured Hypermedia: A Matter of Style.*

2001-6 Martijn van Welie (VU). *Task-based User Interface Design.*

2001-7 Bastiaan Schönhage (VU). *Diva: Architectural Perspectives on Information Visualization.*

2001-8 Pascal van Eck (VU). *A Compositional Semantic Structure for Multi-Agent Systems Dynamics.*

2001-9 Pieter Jan 't Hoen (UL). *Towards Distributed Development of Large Object-Oriented Models: Views of Packages as Classes.*

2001-10 Maarten Sierhuis (UvA). *Modeling and Simulating Work Practice - BRAHMS: A Multiagent Modeling and Simulation Language for Work Practice Analysis and Design.*

2001-11 Tom van Engers (VU). *Knowledge Management: The Role of Mental Models in Business Systems Design.*

2002-1 Nico Lassing (VU). *Architecture-level Modifiability Analysis.*

2002-2 Roelof van Zwol (UT). *Modelling and Searching Web-based Document Collections.*

2002-3 Henk Ernst Blok (UT). *Database Optimization Aspects for Information Retrieval.*

2002-4 Juan Roberto Castelo Valdueza (UU). *The Discrete Acyclic Digraph Markov Model in Data Mining.*

2002-5 Radu Serban (VU). *The Private Cyberspace: Modeling Electronic Environments Inhabited by Privacy-concerned Agents.*

2002-6 Laurens Mommers (UL). *Applied Legal Epistemology: Building a Knowledge-based Ontology of the Legal Domain.*

2002-7 Peter Boncz (CWI). *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications.*

2002-8 Jaap Gordijn (VU). *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas.*

2002-9 Willem-Jan van den Heuvel(KUB). *Integrating Modern Business Applications with Objectified Legacy Systems.*

2002-10 Brian Sheppard (UM). *Towards Perfect Play of Scrabble.*

2002-11 Wouter Wijngaards (VU). *Agent Based Modelling of Dynamics: Biological and Organisational Applications.*

2002-12 Albrecht Schmidt (UvA). *Processing XML in Database Systems.*

2002-13 Hongjing Wu (TUE). *A Reference Architecture for Adaptive Hypermedia Applications.*

2002-14 Wieke de Vries (UU). *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems.*

2002-15 Rik Eshuis (UT). *Semantics and Verification of UML Activity Diagrams for Workflow Modelling.*

2002-16 Pieter van Langen (VU). *The Anatomy of Design: Foundations, Models and Applications.*

CWI	Centrum voor Wiskunde en Informatica
EUR	Erasmus Universiteit Rotterdam
KUB	Katholieke Universiteit Brabant
TUD	Technische Universiteit Delft
TUE	Technische Universiteit Eindhoven
UL	Universiteit Leiden
UM	Universiteit Maastricht
UT	Universiteit Twente
UU	Universiteit Utrecht
UvA	Universiteit van Amsterdam
VU	Vrije Universiteit